Stanford Artificial Intelligence Laboratory
Memo AIM-337

Computer Science Department
Report No. STAN-CS-80-808

May 1980

LEVEL II

FINAL REPORT

Basic Research in Artificial Intelligence and Foundations of Programming

by

John McCarthy, Principal Investigator
Thomas Binford, David Luckham, Zohar Manna, Richard Weyhrauch
Associate Investigators

Edited by Les Earnest

Research sponsored by

Defense Advanced Research Projects Agency

COMPUTER SCIENCE DEPARTMENT
Stanford University

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: STAN-CS-80-808 (AIM-337)

2. GOVT ACCESSION NO.: AD-A091183

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): FINAL REPORT
Basic Research in Artificial Intelligence and Foundations of Programming

5. TYPE OF REPORT & PERIOD COVERED: technical, May 1980

6. PERFORMING ORG. REPORT NUMBER: STAN-CS-80-808 (AIM-337)

7. AUTHOR(s): John McCarthy (Principal Investigator), Thomas Binford, David Luckham, Zohar Manna, and Richard Weyhrauch (Associate Investigators)

8. CONTRACT OR GRANT NUMBER(s): MDA 903-76-C-0206

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Department of Computer Science
Stanford University
Stanford, California 94305 USA

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Avenue, Arlington, Virginia 22209

12. REPORT DATE: May 1980

13. NO. OF PAGES: 75

14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office):
Mr. Philip Surra, Resident Representative
Office of Naval Research, Durand 165
Stanford University

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Recent research results are reviewed in the areas of formal reasoning, mathematical theory of computation, program verification, and image understanding.

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Stanford Artificial Intelligence Laboratory
Memo AIM-337

May 1980

Computer Science Department
Report No. STAN-CS-80-808

FINAL REPORT

Basic Research in Artificial Intelligence and Foundations of Programming

by

John McCarthy, Principal Investigator
Thomas Binford, David Luckham, Zohar Manna, Richard Weyhrauch
Associate Investigators

Edited by Les Earnest

Abstract

Recent research results are reviewed in the areas of formal reasoning, mathematical theory of computation, program verification, and image understanding.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2494, Contract MDA903-76-C-0206. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, or any agency of the U. S. Government.

Reproduced in the U.S.A. Available from the National Technical Information Service, Springfield, Virginia 22161.

Table of Contents

1. Introduction

This report describes recent research in several related areas.

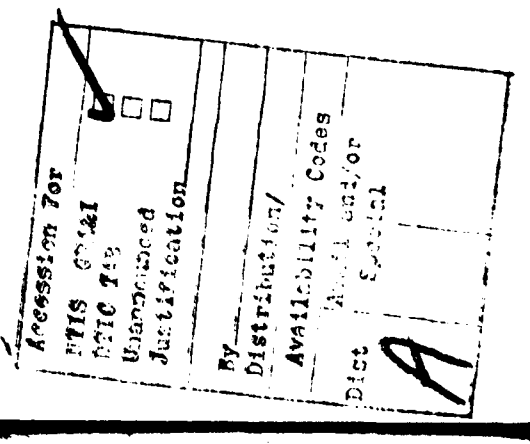
- *Basic research in artificial intelligence and formal reasoning* addresses fundamental problems in the representation of knowledge and reasoning processes applied to this knowledge. Solution of these problems will make possible the development of analytical applications of computers with large and complex data bases, where current systems can handle only a very restricted set of data structures and queries.

- *Mathematical theory of computation* studies the properties of computer programs. The goal is to provide a sound theoretical basis for proving correctness or equivalence of designs.

- The goal of *program verification* is to improve the reliability of important classes of programs such as compilers, operating systems and realtime control systems, and to standardize techniques for program construction, documentation and maintenance.

- *Image understanding* is aimed at mechanizing visual perception of three-dimensional objects either from photographs or from passive imaging sensors. Advances in this field are expected to lead to much more efficient photointerpretation capabilities as well as automatic visual guidance systems.

Appendices list dissertations, films, and other recent reports as well as recent external publications by the staff.



2. Basic Research in Artificial Intelligence and Formal Reasoning

Personnel: John McCarthy,
Richard Weyhrauch, Lewis Creary,
Luigia Aiello, Jussi Ketonen, Ma Xiwen
Student Research Assistants:
Christopher Goad, Carolyn Talcott,
Ben Moszkowski

Applied research requires basic research to replenish the stock of ideas on which its progress depends. The long range goals of work in basic AI and formal reasoning are to make computers carry out the reasoning required to solve problems. Our work over the past few years has made it considerably clearer how our formal approach can be applied to AI and MTC problems. This brings application nearer, especially in the area of mathematical theory of computation.

2.1 Formal Reasoning

John McCarthy continued his work in formalization of the common sense facts about the world needed for intelligent behavior and also work in formalization of computer programs. This work led to the following reports and publications:

1. Two papers with R.S. Cartwright on the representation of recursive programs in first order logic were published in POPL conferences [2, 3].

2. In Spring 1979 McCarthy discovered a formalism for representing sequential programs in first order logic now called Elephant - it never forgets. Unlike previous formalisms, Elephant uses time explicitly as an independent variable and a program counter as an explicit function of time. As a result, properties of programs can be proved directly from the program and the axiomatization of the data domain. Besides that, Elephant programs can refer directly to past inputs and events - sometimes obviating the need for the programmer to define certain data structures for remembering these events; the data structures are then supplied by the compiler. In this one respect Elephant is a higher level language then any developed so far.

3. Work continued on formalisms for representing knowledge. Ma Xiwen revised McCarthy's set of axioms for representing the rather difficult knowledge puzzle of Mr. S and Mr. P and used FOL to verify the translation of the knowledge problem to a purely arithmetic problem. This work is part of a study of the representation of facts about knowledge that will aid the development of programs that know what people and other programs do and don't know.

4. McCarthy further developed his method called circumscription of non-monotonic reasoning and related it to the methods developed by Raymond Reiter and by Drew McDermott and Jon Doyle jointly. It turns out that humans do non-monotonic reasoning all the time, and machines must do it in order to behave intelligently. In our opinion, the formalization of non-monotonic reasoning is a major theoretical step preparing for practical AI systems that can avoid excessively specialized formulations of their knowledge of the world.

A conference on non-monotonic reasoning was held at Stanford in November 1978, and its proceedings are being published as a special issue of the journal *Artificial Intelligence*.

Carolyn Talcott continued her studies in preparation for writing a dissertation that will combine the methods of recursive function theory with the programming techniques of LISP. Hopefully, it will permit the formulation and proof of more complex facts about more complex programs.

Lewis Creary continued his work on the epistemology of artificial intelligence, and on the design of an advice-taking problem solver.

On the epistemological side, new ideas were developed for dealing with:

a) causal reasoning and the frame and qualification problems,

b) the formal representation of actions,
c) epistemic feasibility of actions and the need to plan for the availability of knowledge, and
d) the structuring, classification, and use of domain-specific knowledge within a general problem solving system.

Work on the advice-taking problem solver was concentrated for the most part on high-level design issues and associated theoretical questions. A tentative high-level design and long term plan were constructed, incorporating the epistemological ideas mentioned above, as well as new theoretical ideas concerning:
e) problematic goal seeking, goal refinement, and achievement planning
f) advice seeking and taking as a means of program development and selective knowledge acquisition,
g) complex motivational structures for use in problem solving.

References

[1] Cartwright, R.S., *A Practical Formal Semantic Definition and Verification System for Typed Lisp*, *Ph.D. Thesis*, Computer Science Department, Stanford University, Stanford, California, 1976.

[2] Cartwright, Robert and John McCarthy, Recursive Programs as Functions in a First Order Theory, in E. Blum and S. Takasu (eds.), *Proceedings of the International Conference on Mathematical Studies of Information Processing*, Springer Publishers, 1978.

[3] Cartwright, Robert and John McCarthy, First Order Programming Logic, *Proc. 6th Annual ACM Symp. on Principles of Programming Languages*, San Antonio, Texas, January 1979.

2.2 First Order Logic

During the past year FOL group efforts have been devoted mainly to the following activities: 1) writing and giving talks about projects; 2) doing many new experiments, some of which are described below; 3) planning for the next version of FOL and beginning its implementation (this has taken the form of a new implementation of LISP which is described below); 4) using the Prawitz normalization theorem to automatically specialize bin packing algorithms; 5) the invention (by Ketonen and Weyhrauch) of a new decision procedure for a subset of predicate logic.

1) Publications are listed below. The bulk of the other writing consists of a set of lecture notes by Richard Weyhrauch which he hopes will become a monograph on FOL and is presently about 125 pages long.

2) Weyhrauch is preparing a paper that contains many diverse examples uses of FOL to express many currently interesting problems discussed by AI people. These include non-monotonic logic, morning star/evening star, meta-theory, Dynamic introduction of new reasoning principles, reasoning about belief, Procedural vs. Declatative representation, Not vs. Thnot, and reasoning about inconsistencies. These examples will demonstrate the breath of the FOL epistemology. In addition Luigia Aiello did extensive work demonstrating how metatheory can be used to represent elementary algebra in a reasonable way. This work will be presented at this years automatic deduction workshop.

3) Planning for the next version of FOL has taken several directions. The first amounts to discussions about what the underlying logic of the next version should look like. The second involves how to integrate the work of Chris Goad involving normalization into the new FOL system. The third is what kind of programming language should FOL be written in. The most important issue here is how we intend to implement the FOL evaluator in such a way that the FOL code becomes self

axiomatizing and we can use the FOL evaluator itself as our basic interpreter. This discussion is central to Carolyn Talcott's thesis. This has resulted in Weyhrauch producing a new assembly language programmed version of LISP which is designed to be theoretically smooth enough that an axiomatization of its properties can be carried out by Carolyn Talcott.

4) The work of producing a computationally efficient implementation of normalization was completed early in the year and a series of experiments whose purpose is to assess the practical applicability of proofs as descriptions of computation was carried out. Chris Goad has chosen a class of bin packing problems as a test domain. The work involved both building up a collection of proofs which established some computationally useful facts about the domain, and investigating the computational efficiency aspects of these proofs. This work has resulted in Chris finishing his Ph. D. thesis which he is currently writing up.

5) Ketonen worked out details and greatly extended an idea of Weyhrauch's about the possibility of a new decision procedure for a subset of the predicate calculus they are now calling the direct part of first order logic. This algorithm is currently being coded by Ketonen.

Publications

[1] L. Aiello and G. Prini, An efficient interpreter for the LAMBDA calculus, to be published in *Journal of Computer and System Sciences*, 1979.

[2] G. Prini, Applicative parallelism, submitted for publication in *Journal of the ACM*, 1979.

[3] L. Aiello, and G. Prini, Design of a personal computing system, to be published in *Proceeedings of the Annual Conference of the Canadian Information Processing Society*, Victoria (British Columbia), May 12-14, 1980.

[4] L. Aiello, and G. Prini, Operators in LISP: some remarks on Iverson's Operators, to be published in *ACM Transactions on Programming Languages and Systems*, 1980.

[5] G. Prini, Explicit parallelism in LISP-like languages, submitted to 1980 LISP Conference, Stanford University, 24-27 August 1980.

[6] L. Aiello, Evaluating functions defined in first-order logic (tentative title), submitted to Logic Programming Conference, Budapest, July 1980.

[7] L. Aiello, R. Weyhrauch, Using meta-theoretic Reasoning to do Algebra, *Proc. Automatic Deduction Conference*, Les Arcs (France), July 8-11, 1980.

[8] R. Weyhrauch, Prolegomena to a theory of mechanized formal reasoning, To appear in *AI Journal* special issue on non-monotonic logic.

[9] C. Goad, Proofs as descriptions of computations, *Proc. Automatic Deduction Conference*, Les Arcs (France), July 8-11, 1980.

2.3 Mathematical Theory of Program Testing

This thesis by Martin Brooks describes how testing can be used to show a program's correctness. The scenario for using testing to show correctness is that the programmer writes a program P which is either correct, or which differs from the correct program by one or more erors in a specified class E of errors. The theory relates:

(1) P's structure.
(2) The combinatorics of how errors in E could have led to P (incorrectly) being written.
(3) The type of program behaviour that the programmer will observe when testing, for example the program's output, or a trace.
(4) The characterisitics of "complete" test data, which is guaranteed to reveal the presence of errors in E, assuming all errors come from E.

The theory is developed at a general level and is then specialized to the case of recursive programs having only simple errors, where the programmer observes a trace of each test run. A method is devised for generating test data, based on this application of the theory. The generated test data is "complete", meaning that if the only errors in the recursive program are of the specified types, then the test data will reveal the errors through at least one incorrect trace.

Viewed contrapositively, assuming that the written program P is "almost correct", that is, differs from the correct program at most by errors of the specified simple types E, if the trace of each test run is as the programmer intended, then the program must be correct. The test data generation method has been implemented (in MacLisp) and examples of its operation are given.

Reference

[1] Martin Brooks, Determining Correctness *by testing*, *Stanford AI Memo* AIM-336, June 1980.

2.4 Proofs as Descriptions of Algorithms

Chris Goad is currently completing his Ph.D. thesis, titled "Computational uses of the manipulation of formal proofs". The thesis concerns the use of proofs as descriptions of algorithms. In particular, a proof in an appropriate system of a formula of the form "for all x there exists a y such that R(x,y)" can be used express an algorithm A which satisfies the "specification" R in the sense that, for each input n, R(n,A(n)) holds. The proof can be "executed" by use of any of several methods derived from proof theory, for example, by a proof normalization procedure, or by extraction of "code" from the proof by means of one of the functional interpretations.

A proof differs from a conventional program expressing the same algorithm in that it formalizes more information about the algorithm than does the program. This additional information expands the class of transformations on the algorithm which are amenable to automation. For example, there is a class of "pruning" transformations which improve the computational efficiency of a natural deduction proof by removing unneeded case analyses. These transformations make essential use of dependency information which finds formal expression in a natural deduction proof, but not in a conventional program. Pruning is particularly useful in adapting general purpose algorithms to special situations.

Goad has developed efficient methods for the execution and transformation of proofs, and has implemented these methods on the Stanford Artificial Intelligence Laboratory's PDP-10 computer. He has done a series of experiments on the specialization of a bin-packing algorithm. In these experiments, substantial increases of efficiency were obtained by use of the additional "logical" information contained in proofs.

The general objective of Goad's work has been the development of improved technology for manipulating methods of computation. In his thesis, he has exhibited new automatic operations on algorithms - operations which are made possible by the use of formal proofs to describe computation.

Reference

[1] C. Goad, Proofs as descriptions of computations, *Proc. Automatic Deduction Conference*, Les Arcs (France), July 8-11, 1980.

3. Mathematical Theory of Computation

Personnel: Zohar Manna.

(a) A Deductive Approach to Program Synthesis (references [2,3,6])

A framework for program synthesis has been developed that relies on a theorem-proving approach. This approach combines techniques of unification, mathematical induction, and transformation rules within a single deductive system. We investigated the logical structure of this system and considered the strategic aspects of how deductions are directed. Although no implementation exists, the approach is machine oriented and ultimately intended for implementation in automatic synthesis systems.

(b) The Modal Logic of Programs (reference [5])

We explored the general framework of Modal Logic and its applicability to program reasoning. In relating the basic concepts of Modal Logic to the programming environment, we adopted the temporal interpretation of Modal Logic: the concept of "world" corresponds to a program state, and the concept of "accessibility relation" corresponds to the relation of derivabiliy between states during execution. A variety of program properties expressible within the modal formalism was investigated.

The first axiomatic system studied, the "sometime" system, is adequate for proving total correctness and "eventuality" properties. However, it is inadequate for proving "invariance" properties. The stronger nextime system obtained by adding the next operator was shown to be adequate for invariances as well.

We first applied these systems to the analysis of deterministic programs, then we examined the implications of nondeterminism, and finally investigated the extensions necessary to handle concurrency.

(c) A Situational-Calculus Approach to Problematic Features of Programming Languages (reference [1])

Certain features of programming languages, such as data structure operations and procedure call mechanisms, have been found to resist formalization by classical techniques. We developed an approach, based on a situational calculus that makes explicit reference to the states of a computation. In this calculus, the evaluation of an expression yields a value and produces a new state. For each state, distinction is drawn between an expression, the location of its value, and the value itself.

Within this conceptual framework, many of the "problematic" features of programming languages can be described precisely, including operations on such data structures as arrays, pointers, lists, and records, and such procedure call mechanisms as call-by-value, call-by-reference, and call-by-name. No particular obstacle is presented by aliasing between variables or by recursion in procedure calls. The framework provides an expressive vocabulary for defining the semantics of programming-language features. It is amenable to implementation in program verification or synthesis systems.

(d) Synchronous Schemes and Their Decision Problems (reference [4])

A class of schemes called synchronous schemes was defined. A synchronous scheme may have several variables but requires the values assigned to these variables to be synchronized during all computations. This means that in every symbolic computation the differences between the heights of the Herbrand values of the variables are kept bounded. This boundedness ensures a convergent "annotate and unwind" transformation which yields an equivalent fully annotated free scheme. As a result, the problems of convergence, divergence, equivalence, and other properties are decidable for schemes in this class. Membership in this class for a given bound is decidable. The class can be extended by allowing "reset statements" which cause all the variables to be reset to constant values, as well as dead variables, which from a certain point on remain constant and hence lag behind the other variables.

The class of synchronous schemes contains, as special cases, the known classes of Ianov schemes, one-variable schemes and progressive schemes, for which equivalence and other properties are decidable.

Recent Publications

1. Z. Manna and R. Waldinger, Problematic *Features of Programming Languages: A Situational-Calculus Approach*, in preparation.

2. Z. Manna, Lectures on the Logic of Computer Programming, SIAM Press (Feb. 1980).

3. Z. Manna and R. Waldinger, A Deductive Approach to Program Synthesis, ACM Transactions on Programming, Languages, and Systems, vol. 2, no. 1, (Jan. 1980), pp. 90-121.

4. Z. Manna and A. Pnueli, Synchronous Schemes and Their Decision Problems, Proceedings Seventh ACM Symposium on Principles of Programming Languages, Las Vegas (Jan. 1980), pp. 62-67.

5. Z. Manna and A. Pnueli, A Modal Logic of Programs, Proceedings International Conference on Automata, Languages and Programming, Graz, Austria (July 1979), Invited paper.

6. Z. Manna and R. Waldinger, Synthesis: Dreams→Programs, IEEE Transactions on Software Engineering, vol. SE-5, no. 4, (July 1979).

4. Program Verification

Personnel: David C. Luckham, Derek C. Oppen, Friedrich W. von Henke, Steven M. German, Wolfgang Polak, Richard Karp, Howard Larsen, Peter Milne.

The research of the Program Analysis and Verification Group is directed towards the development of new programming methods and automated programming aids. The goal is efficient production of very reliable systems programs, including compilers and operating systems, and efficient maintenance of such programs.

The group is actively conducting research in three main areas:

1. Design and implementation of interactive program analyzers.

2. Design of a high level programming language and associated specification language for multi-processing. This language is equivalent to an extensive ADA subset but in addition enforces a rigorous specification standard.

3. Application of program analyzers, and particularly verifiers, to such programming problems as debugging, documentation, proof of correctness, and analysis of modifications to code and specifications.

Systems that automate or partially automate the analysis of properties of programs may be collectively named "program analyzers". The group has implemented two analyzers, the Stanford Pascal Verifier and the Runcheck system. The Stanford verifier is intended for interactive use as an aid to checking the correctness of programs. It automates methods for analysing the consistency of a program with its documentation. It is being distributed on a limited basis to other research laboratories for evaluation and for experimental use as a programming aid to debugging and structured programming. Runcheck is a development of the verifier for fully automatic analysis of a program for possible common runtime errors. It automates some simple methods for improving program documentation and analysing why verifications fail.

Our experiments with these analyzers have made it clear that verification methods can be applied to analysis of other programming problems, including adequacy of documentation, efficiency of code, and adaptability of existing code to new specifications.

Currently the group is working towards applying analysis based on verification techniques to a very wide range of programs. These include:
(i). new kinds of programs previously considered to be beyond the limits of present techniques in automated verification; e.g., complicated pointer manipulating programs, and large programs such as a compiler.
(ii). programs using new language constructs including Modules, Concurrent Processes and Message Passing constructs.

This has required a research effort in design of programming languages and specification (or assertion) languages, and in programming methods, particularly in the area of multi-processing. This research is being carried out with very careful attention to the ADA design effort [4,16] and all implementation and verification results will be directly applicable to ADA.

The references contain some of the group's earlier work in areas 1 and 3 (above) which has been published (e.g., [3, 7, 13, 17, 21, 22, 25, 27, 28]). Reference [37] is Edition 1 of the Verifier User Manual.

Accomplishments

1. Verifier System

1.1 The group has implemented a verifier for almost the full Pascal language (exclusions mainly concern floating point arithmetic).

1.2 The user manual [37] contains an introduction to program verification and examples illustrating the use of the verifier as an aid in debugging, documentation and structured programming. The manual has been distributed widely and is now in a second printing.

1.3 The verifier has recently been introduced at several other ARPAnet sites to test its portabilty and to obtain some preliminary feedback from other user groups. A version for use over the net is provided on the Stanford SCORE computer. It has been used over the net by several user groups including CSIRO, Canberra, Australia and the Norwegian Defense Research Establishment, Oslo, Norway. Export versions are being used as a basis for courses in program verification at three or more universities in the USA and two universities in Australia.

1.4 We have begun distribution of the verifier on a limited basis both to government projects interested in security aspects of systems programs, and to industrial research laboratories interested in programming languages, specification languages, software tools, and verification applied to products that have very high reliabilty requirements.

1.5 A library of documented and verified programs is being built and is available over the ARPAnet from SAIL on the directory [EX,VER] and from SU-SCORE on directory <CSD.VER>.

The verifier is an experimental prototype system, and it is hoped that this distribution (1.1 - 1.5 above) will generate feedback essential to motivating development and improvements in automated verifiers and their applications.

2. Specification Languages

The language accepted by the verifier extends Pascal by inclusion of Modules in a form compatible with the DoD Ada language design [16] (see also DoD Steelman[4]).

2.1 A theory of documentation of modules has been worked out (Luckham and Polak [24]). This theory is new and has not been suggested in any other previous language design. Its practicality is being tested. The verifier requires modules to be documented. Verification of documented modules has already been successful on some substantial examples. One example is a module implementing Pascal pointer and memory allocation operations using a subset of Pascal whose only complex data type is Arrays. These extensions will be documented in the next edition of the user manual.

2.2 The theory of Pascal pointer verification (Luckham and Suzuki [25]) has finally been published in TOPLAS, five years after its implementation in earlier versions of the Stanford verifier and four years after the original AI Lab. report. Recent discussions show that some other verification groups are now experimenting with our method of pointer specification and verification by means of Reference classes.

2.3 Specifications for multiprocess systems have been studied and developed. A method of specification and implementable axiomatic semantics of concurrent processes has been given in [23], and also in Karp's Ph.D. thesis. This study is expected to contribute toward the development of documentation standards and automated verifiers for Ada.

3. Runcheck and Programming Language design

3.1 A special version of this verifier for automatic detection of runtime errors in programs has been implemented. This is called the Runcheck system. Results with an early version of Runcheck have been published

(German, [7]), and are the most impressive in the area of completely automatic analysis of programs so far. A more comprehensive report on this system is forthcoming [9]. Work on improvements to this system and on automatic documentation of Pascal and Ada programs is continuing.

3.2 Recently, work on this area has also focused on the relationship between verification and language design. For instance, a secure verifiable Union type to replace Pascal's variant records has been designed and formally specified. Checking for runtime errors involving unions is a straightforward verification problem.

3.3 Certain very efficient operations have traditionally been excluded from high level languages because errors with them would have serious and confusing side effects. Work has begun on the idea that some of these operations, such as explicit deallocation for dynamic variables and conversion between arbitrary types, can be safely included in a high level language by verifying that they are used correctly.

4. Theorem Provers and Rule Languages.

4.1 The success of these analyzers depends on recent advances made by this group in the theory and implementation of cooperating special purpose decision procedures (Nelson and Oppen [27, 28], Oppen [29, 30]). This new approach to implementing theorem provers is the best method found to date. The present prover automatically handles any quantifier-free formula over the integers, arrays, list structure and Pascal records. It has automatically proved formulas over 300 lines long which have been produced by the RUNCHECK version of the verifier.

4.2 In developing an underlying theorem proving capability a second important area of progress has been the design and implementation of a Rule Language for user-supplied lemmas. This latter facility allows the user to describe data structures not handled by special purpose provers and to define specification languages, i.e. new concepts used in the specification of programs. Its implementation, the "rulehandler", gives the verifier the ability to use definitions of specification concepts in correctness proofs. Details of the present rulehandler are in the user manual [37] and in [35].

4.3 Experiments have been made in which a specification language for a particular class of programs (e.g., simple database programs) was defined by a set of 20 rules; using this set of rules, the verifier was able to carry out correctness proofs of programs that required proving over 120 verification conditions. It would be completely impractical to carry out such proofs by hand (and exhaustive testing of such programs would also be very difficult). The rule language provides the means of describing the basic knowledge necessary for automatically proving a large number of different (but similar) programs. The design of rule languages is still very much a research area, and we are continuing research to improve the present facility.

5. Verification Experiments

5.1 A number of programs operating on binary trees has been verified as part of a study on top-down development and verification of programs involving manipulation of pointer structures. First results will be reported in [12]. The verified programs constitute the main part of a program implementing the fastest known algorithm for merging balanced binary trees.

5.2 An in-place permuation algorithm presented by D. Knuth in 1971 has been successfully verified. In 1971, D. Knuth claimed that this algorithm was intractable to automatic verification at that time. A paper [32] describing this proof has been published in IEEE Transactions on Software Engineering.

5.3 In connection with the compiler verification project [34] (described in more detail in Section 2.4) several fairly large proofs have been completed. A scanner, a parser, a type checking

program, and a code generator for a Pascal like language have been verified. These programs total 150 pages of Pascal code; they may well be the largest known set of programs to be verified with the aid of a verification system at this point. Also the verification conditions are probably the largest formulas ever succesfully treated by an automatic theorem prover.

References

[1] Brinch-Hansen, P., The Solo Operating System: A Concurrent Pascal Program, *Software — Practice and Experience* 6, 1976.

[2] Bochmann, G., and Gecsei, J., A Unified Method for the Specification and Verification of Protocols, *Proceedings IFIP Congress 1977*, pp.229-234.

[3] Cartwright, R. and Oppen, D., Unrestricted Procedure Calls in Hoare's Logic, to appear, *Acta Informatica*. (Also *Proc. 5th ACM Symposium on Principles of Programming Languages*, ACM, New York, 1978).

[4] DoD., Requirements for High Order Computer Programming Languages, Department of Defense revised Steelman, July 1978.

[5] Drysdale, R.L., and Larsen, H.J., Verification of sorting programs, AI Lab. Memo, forthcoming.

[6] Flon, L. and Suzuki, N., Nondeterminism and the Correctness of Parallel Programs, *Proc. IFIP Working Conference on the Format Description of Programming Concepts*, St. Andrews, New Brunswick, 1977.

[7] German, S., Automating Proofs of the Absence of Common Runtime Errors, *Proc. 5th ACM Symposium on Principles of Programming Languages*, ACM, New York pp.105-118, 1978.

[8] German, S., Luckham, D.C., and Oppen, D., Proving the absence of common runtime errors, draft Stanford A.I. Lab. Memo, Stanford University, forthcoming.

[9] German, S., An Automatic System for Proving the Absence of Common Runtime Errors, Stanford A.I. Lab. Memo, Stanford University, forthcoming.

[10] German, S., Verification with Variant Records, Unions Types, and Direct Access to Data Representations, Stanford A.I. Lab Memo, Stanford University, in preparation.

[11] Haynes, G., Verification Issues in an Air Traffic Control System, Texas Instruments Inc. Report, 1979.

[12] v. Henke, F.W., Top-down development and verification of programms, Stanford A.I. Lab. Memo, forthcoming.

[13] v.Henke, F.W. and Luckham, D.C., A methodology for verifying programs, *Proc. Int. Conf. on Reliable Software*, Los Angeles, California, April 20-24, 1975, 156-164.

[14] v. Henke, F.W. and Luckham, D.C., Verification as a Programming Tool, Stanford A.I. Lab. Memo, forthcoming.

[15] Hoare, C.A.R., Monitors: An Operating System Structuring Concept, *CACM* 17: 10 (1974), pp 549 – 557.

[16] Ichbiah, J.D., Krieg-Brueckner, B., Wichmann, B.A., Ledgard, H.F., Heliard, J-C., Abrial, J-R., Barnes, G.P., and Roubine, O., Reference Manual for the Green Programming Language, CII Honeywell Bull, March 1979.

[17] Igarashi, S., London, R.L., and Luckham, D.C., Automatic program verification I: Logical basis and its implementation, *Acta Informatica*, vol.4, 1975, 145-182.

[18] Jensen, K. and Wirth, N., *Pascal User Manual and Report*, Springer-Verlag, New York (1975).

[19] Kahn, G., The Semantics of a Simple Language for Parallel Programming, *Proc. IFIP*, North-Holland Publishing Company, Amsterdam, 1974.

[20] Kahn, G. and MacQueen, D., Coroutines and Networks of Parallel Processes, *Proc. IFIP*, North-Holland Publishing Company, Amsterdam, 1977.

[21] Karp, R., and Luckham, D., Verification of Fairness in an Implementation of Monitors, *Proc. 2nd Int. Conf. on Software Engineering*, San Francisco, 1976.

[22] Luckham, D.C., Program Verification and Verification-Oriented Programming, *Proc. IFIP Congress 77*, pp. 783-793, North-Holland publishing Co., Amsterdam, 1977.

[23] Luckham D.C., and Karp, R.A., Axiomatic Semantics of Concurrent Cyclic Processes, Stanford Verification Group report, forthcoming; submitted for publication.

[24] Luckham, D. and Polak, W., A Practical Method of Documenting and Verifying Programs with Modules, draft manuscript, A.I. Lab. Memo, forthcoming, submitted for publication (1978),

[25] Luckham, D.C., and Suzuki, N., Automatic program verification IV: Proof of termination within a weak logic of programs, Stanford AI Memo AIM-269, Stanford University, 1975, *Acta Informatica*, vol. 8, 1977, 21-36.

[26] Luckham, D.C., and Suzuki, N., Verification oriented proof rules for arrays, records, and pointers. Stanford Artificial Intelligence Project Memo 278, Stanford University, April 1976; published as: Verification of Array, Record, and Pointer Operations in Pascal, ACM Transactions on Programming Languages and Systems, Oct. 1979, pp. 226-244.

[27] Nelson, C. and Oppen, D., Fast Decision Procedures based on Congruence Closure, to appear *JACM*. (Available as CS Report No. STAN-CS-77-646, Stanford University, or in *Proc. 18th Annual IEEE Symposium on Foundations of Computer Science*).

[28] Nelson, C. and Oppen, D., Simplification by Cooperating Decision Procedures, *ACM Transactions on Programming Languages and Systems*, Oct. 1979. (Available as CS Report No. STAN-CS-78-652, Stanford University, or in *Proc. 5th ACM Symposium on Principles of Programming Languages*, ACM, New York.)

[29] Oppen, D., Reasoning about Recursively Defined Data Structures, to appear *JACM*. (Available as CS Report STAN-CS-78-678, Stanford University, or in *Proc. 5th ACM Symposium on Principles of Programming Languages*, ACM, New York.)

[30] Oppen, D., Convexity, Complexity, and Combinations of Theories, to appear, Theoretical Computer Science. *Proc. 5th Symposium on Automated Deduction*, January 1979.

[31] Owicki, S., Specifications and Proofs for Abstract Data Types in Concurrent Programs, Technical report 133, Digital Systems Laboratory, Stanford, 1977.

[32] Polak, W., An exercise in automatic program verification, *IEEE Trans. on Software Engineering*, vol. SE-5, Sept. 1979.

[33] Polak, W., Program verification based on denotational semantics, Stanford Verification Group Report, forthcoming.

[34] Polak, W., Theory of Compiler Specification and Verification, PhD Thesis, Stanford University, forthcoming.

[35] Scherlis, W., A Rule Language and its Implementation, in preparation, A.I. Lab Memo, forthcoming; submitted to the AI journal.

[36] Scott, D., Outline of a Mathematical Theory of Computation, Technical monograph PRG-2, Oxford University Computing Laboratory, Oxford, England (1970).

[37] Stanford Verification Group, Stanford Pascal Verifier User Manual, Edition 1, Stanford Verification Group Report No. 11, Report STAN-CS-79-731, Stanford University, March 1979.

[38] Suzuki, N., Automatic Verification of programs with complex data structures, Ph.D. thesis, Computer Science Department, Stanford University, 1976.

[39] Wirth, N., MODULA: A language for modular multiprogramming, ETH Zurich, 1976.

5. Image Understanding

Personnel: Thomas Binford, Barry Soroka, *Student Research Assistants:* Donald Gennery, Reginald Arnold, Rodney Brooks, Hans Moravec.

The program of research of the Image Understanding group (SAIL-IU) is leading to implementing powerful computer vision systems which perform the following tasks in Photointerpretation, Guidance, and Cartography:

- monitor aircraft at airfields;
- monitor changes in building complexes;
- locate vehicles in parking lots and off roads from aerial photos;
- reconstruct geometry of building complexes from stereo images to make reference images;
- navigate at low altitude with passive sensors.

SAIL-IU capabilities will contribute to the IU-DMA Testbed.

These scenarios cover important tasks which include a broad variety of object classes (aircraft, airfields, buildings, terrain, vehicles) and a range of imaging conditions (high and low altitude, a variety of sensors, ranges of illumination, weather conditions, snow and ground cover). The scenarios provide a test of generality and relevance.

SAIL-IU has made significant progress in several areas of Image Understanding during the contract period:

1. development of ACRONYM, an advanced model-based interpretation system;
2. first demonstration of its interpretation system, ACRONYM, in recognition of a Lockheed L1011 in photos of San Francisco airport;
3. progress in algorithms for a high performance stereo vision system;
4. development of an algorithm for finding "ribbons" in images;
5. progress toward a new algorithm for finding edges in images.

SAIL-IU research has found applications in the Passive Navigation Project and in the Autonomous Terminal Homing program.

5.1 Summary

SAIL-IU research has concentrated on building a powerful interpretation system (ACRONYM) including a stereo vision system. ACRONYM integrates a wide range of image understanding capabilities including those from other laboratories.

ACRONYM has successfully identified an L1011 in an aerial photograph of San Francisco airport [Brooks 79c]. Figure 1 shows ACRONYM's model of an L1011. Figure 2 shows an aerial photograph of an L1011 at a passenger terminal at San Francisco airport. Figure 3 shows an edge description of the airport scene. Figure 4 shows large "ribbons" described by a "ribbon" finder. On the basis of these ribbons, a tentative identification was made. Figure 5 shows the result of a search for smaller features which verify the identification; these smaller features are horizontal stabilizers and engine pods. Identification of the aircraft would be significant if it were done by a special purpose program to identify aircraft. The fact that it was done automatically from models by a general purpose model-based vision system makes it a special accomplishment. Implementation of ACRONYM is far along. Relatively powerful versions of all major components exist.

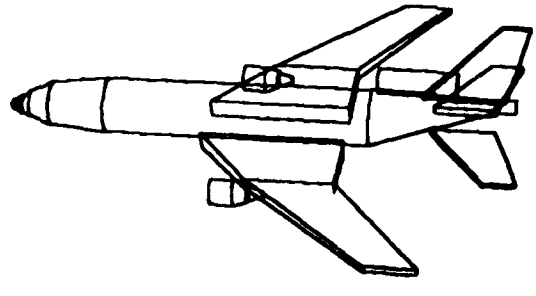


Figure 1

Figure 2

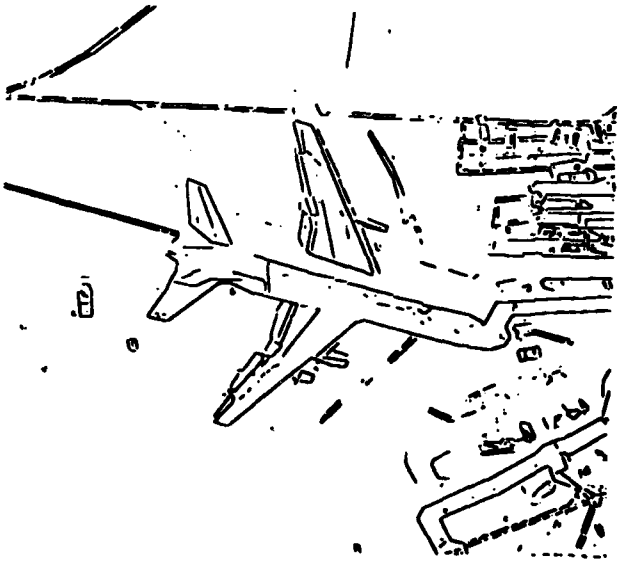


Figure 3

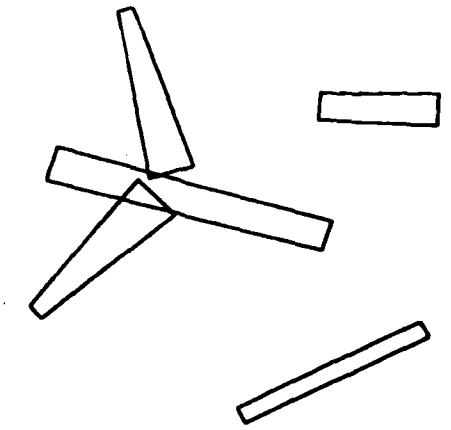


Figure 4

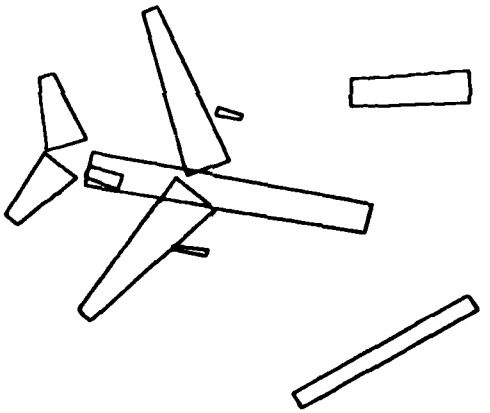


Figure 5

ACRONYM contains subsystems for *geometric modeling, geometric reasoning, prediction, planning, description, and interpretation.* Geometric modeling produces models of objects as Object Graphs (figure 1). The geometric modeling system uses generalized cylinder primitives which provide compact representation of complex shapes including *curved surfaces. It produces symbolic and* generic models needed for geometric reasoning. The modeling system has a geometric editor to aid in constructing models. Geometric reasoning is used in *prediction, planning, description, and interpretation* subsystems. Geometric reasoning is implemented as a rule-based system which uses Object Graphs, Observability Graphs, and Picture Graphs which represent models, expectations, and image descriptions.

Prediction is the process of going from symbolic models to signals or to symbolic descriptions closely derived from signals. Prediction produces Observability Graphs which represent symbolic summaries of expected observables *produced by "ribbon" finders, edge finders,* surface and shape descriptor routines. The most useful predictions are quasi-invariant observables, valid for a wide range of viewpoints and valid for classes of objects, i.e. generic with respect to object class and viewpoint.

Interpretation is the process of mapping descriptions of images and surfaces (Picture Graph) to predictions of appearances (Observability Graph) and to objects (Object Graph). *Description is the process of going* from signal to symbol, from images to edges, ribbons, and surfaces.

A key part of any system is the *descriptive component, the subsystem which obtains* "ribbon", edge, surface and shape descriptions from images and sequences of images. The edge finder of Nevatia and Babu of USC was transported to SAIL at considerable effort [Nevatia and Babu 78]. Its output alone was not sufficient, but an algorithm was developed for linking edge elements into well-formed regions [Brooks 79b]. Edges were linked by a best first search, with heuristics to select candidate edges for linking and to prune the search tree. Meanwhile, an extension is underway of the techniques in the Binford-Horn edge finder [Horn 72].

Development of stereo vision has progressed *rapidly. A new stereo mapping subsystem has* been developed which generates depth maps which are much more complete and more accurate than previous systems [Baker 80]. It was preceded by two earlier stereo mapping *systems [Gennery 77], [Arnold 78]. Meanwhile,* algorithms are being developed for a new stereo mapping system. Significant results have been obtained for geometric constraints on correspondence in stereo images which will lead to powerful and robust stereo matching capabilities.

5.2 Research Objectives

The Image Understanding program aims to contribute to powerful techniques in support of *these missions: 1. Object classification in* photointerpretation and photointelligence in the Image Understanding Testbed, in cooperation with Defense Mapping Agency, relevant to DMA activities. 2. Stereo reconstruction for cultural areas for the Testbed. 3. Stereo reconstruction for making reference maps of cultural areas in support of Autonomous Terminal Homing; stereo image matching and stereo TERCOM (terrain comparison) for guidance in terminal homing. 4. Tracking of linear features and stereo image matching in cultural areas, for navigation with passive sensors.

Technologies which are essential for those missions are: object classification, stereo *mapping of cultural areas, stereo matching, and* linear feature description. A later section describes scientific problems which are central to those techniques. Interpretation and object classification is an enormous problem to which both predictive and descriptive techniques contribute. Both are being integrated in ACRONYM. Both stereo surface features and

linear image feature description are important. While adequate techniques exist for stereo mapping of terrain, stereo mapping for buildings presents difficult problems. Linear feature description is important in its own right, and also supports the other techniques. Recent progress with linear feature description promises significant payoff for all parts of automated photointerpretation. The following paragraphs describe photointerpretation scenarios.

Photointerpretation

In a typical scenario, a photointerpreter (PI) instructs ACRONYM to monitor selected airfields frequently and report number and type of aircraft. He names models of aircraft and airfields from a database.

ACRONYM monitors building complexes in order to alert a PI to changes which might be significant, while ignoring insignificant changes. A PI selects a building complex to be monitored. ACRONYM builds a depth map and spatial model of the complex. The PI labels some objects in the image as buildings, oil tanks, radar antennas, etc., and helps the system augment or improve its model. ACRONYM incorporates results of stereo mapping of new coverage in its model (sides seen from different views, different shadow clues). ACRONYM maps depth changes and interprets them. The PI instructs ACRONYM to ignore vehicle motion, painting, snow, rain, and vegetation surface changes. ACRONYM alerts the PI when depth changes are observed and presents an interpretation, with a registered stereo pair showing changes in depth.

A PI instructs ACRONYM and the Stanford Stereo System to cue on raised objects which stand above surrounding terrain, or to cue on cultural objects.

ACRONYM monitors an area for location, number and types of new buildings. ACRONYM monitors a region for offroad staging areas for vehicles. It reports number and type of vehicles and their activities.

Traditionally, DMA has produced maps and charts. More and more, DMA needs to provide digital data bases and to provide individualized information to particular users. DMA seeks quicker response, higher throughput, better response to user needs, and greater availability of data. Sensing technology provides a flood of image input which current operations cannot handle within reasonable costs. Recording and presenting results of interpretation pose great problems. Tactical situations require quick turnaround which is difficult with the current system. DMA plans to change to a digital system to meet these demands. Much of the sensor data will be digital directly. The Image Understanding Testbed and this research are expected to contribute to the demonstration of candidate capabilities and to aid in the evaluation process which determines the design of future DMA systems.

SAIL-IU research is expected to contribute to developing interactive aids which enhance productivity of PIs by taking over a burden of routine cueing, screening, monitoring, measuring, counting, and recording. Automated systems offer the possibility of extensive, continuous observation of tactical situations, with instantaneous reports.

Passive Navigation and Terminal Homing

SAIL-IU has a subcontract to Lockheed on navigation of low-flying autonomous aircraft with passive sensors. Passive image sensing will improve survivability, provide a capability to fly a flexible course and to navigate with high accuracy with small memory requirements. The approach is intended to make navigation insensitive to change and errors in its reference data base.

SAIL-IU is preparing to work with the Autonomous Terminal Homing program in development of improved stereo compilation algorithms, in assessment and integration of stereo algorithms from the Image Understanding community, and in working with contractors of the terminal homing

program. One terminal homing contractor acknowledges a major contribution to their work from the Image Understanding program.

Scientific Objectives

A photointerpreter solves a puzzle by piecing together clues from a history of images and background information. He relies heavily on spatial information from stereo imaging and shadows, and spatial knowledge about structures, together with collateral reports about structures. The interpretation system should provide these three capabilities: interpreting spatial structures, using multiple cues, and stereo mapping. The SAIL-IU approach is that multiple cues and collateral reports provide knowledge at different levels about geometric structures, and that using multiple cues requires reasoning among different levels of spatial structure.

The interpretation system should be **generalizable** in the following sense: interpretation systems for similar tasks should use the same basic system with different models; interpretation systems for widely different tasks should be assembled from a core of common modules and a few modules which are task specific. Algorithms should solve well-defined generic subproblems which are common to a wide range of interpretation tasks. ACRONYM uses modules based on a hierarchy of representations: context level, object level, volume level, surface level, image structure level, and image level. See figures 6 and 7. These representations and the maps between them are the basis for a vision language. The scenarios outlined above cover a broad range of objects and contexts which will help to develop a general system and maintain generality.

The interpretation system should have a **natural interface** which make it natural and easy for a PI to set up a new interpretation task. The PI should be able to program a new task in a week without expertise as a vision programmer. Users and ACRONYM communicate in a common language of geometric models, in a high level modeling language. Users supply models of object shape and surface descriptors. The high level modeling language provides a bridge to natural language. SAIL-IU aims to provide the underlying geometric representation of shape and spatial relations upon which to build a linguistic system, but not to work on natural language for geometric description. That is better left to those working in natural language.

The system should have **generic interpretation**, generic with respect to object class and generic with respect to viewing conditions. That is, it should represent and recognize object classes, e.g. jet passenger aircraft, and should deal with ranges of viewing conditions, e.g. variations in viewing position and orientation, object orientation, illumination, sensors, and surface markings. It is important information that different configurations of the same aircraft are closely related, instead of separate, unrelated objects.

Summary

ACRONYM includes the following mechanisms:

- segmented part/whole representations of generic parts;
- interpretation at the level of volumes;
- coarse-to-fine interpretation for efficiency;
- a clear representation hierarchy;
- rule-based spatial reasoning;
- a high level modeling language.

Figure 6. Representation Levels

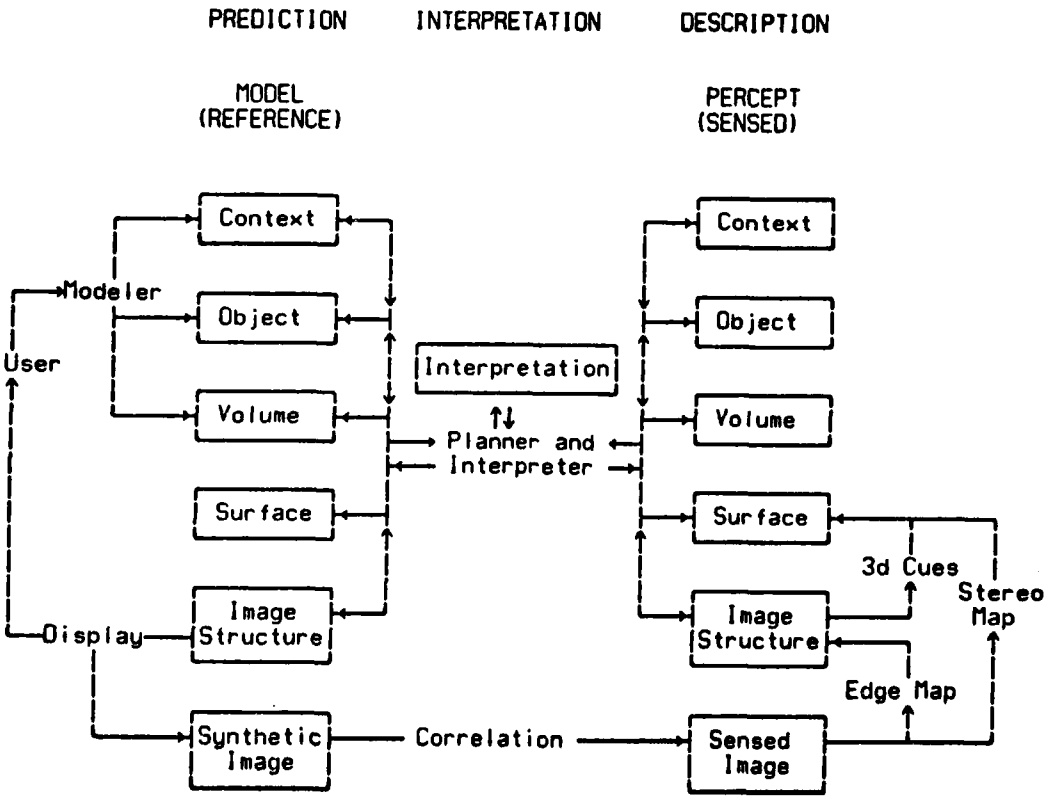


Figure 7

Representations and Levels

LEVEL	REPRESENTATION
Context	Objects
Object	Volumes, Surfaces, Functions
Volume	Generalized Cones, Spheres, Surfaces
Surface	Ribbons
Image Structure	Ribbons, Curves, Texture Vertices
Image *Features*	Edge elements, Spots
Image	Pixels

Generic interpretation poses a major conceptual challenge. Traditional classification of individual objects is not plausible for cueing and screening buildings and vehicles. In traditional classification, each configuration of a truck with load would be a separate template. It is difficult to anticipate all loads that a truck might carry. Buildings are made up of combinations of basic structure elements; for an unstructured matching approach the complexity is combinatorial in the components, while for a segmented, structured matching approach, complexity is additive. The required generalization is to go from two dimensional to three dimensional representations, from unstructured to structured, segmented matching, and to use generic primitives.

Interpretation for generic viewing conditions is equally important. Volume parts of an object are invariant with respect to viewing conditions, but there is no invariance at the image level. One traditional classification approach is to have a dense sampling of separate views of an object. ACRONYM interprets at the level of *volume parts* rather than at the image level. This means that parts of an image need to have the same *three-dimensional interpretation*, not the same appearance, thus images can vary widely.

Efficiency

ACRONYM uses a coarse-to-fine interpretation mechanism which first matches large, obvious parts, then finer detail. The part/whole representation embodies a hierarchy of detail, from coarse to fine. For example, an aircraft has fuselage and wings at the top level; attached to them are engine pods, tail and horizontal stabilizer. ACRONYM uses this inherent structure as a guide to efficient strategies for identification. It is planned to incorporate a set of rules for efficient ordering of evaluation.

5.3 ACRONYM progress

ACRONYM integrates powerful shape representation with rule-based geometric reasoning in an inherently three-dimensional system. ACRONYM has a geometric reasoning capability based on a rule-based backward-chaining production rule system modeled after MYCIN [Davis 1975]. Contact with the MYCIN group has contributed enormously to this effort. ACRONYM has a number of new capabilities which are advances over previous vision systems. ACRONYM uses three dimensional models and matches at the level of three dimensions. Previous vision systems have been quasi-two-dimensional and have largely interpreted at the level of images [Barrow 76], [Kanade 77]. ACRONYM makes strong use of image shape in the form of ribbons and relations between them in selecting initial candidates for interpretation; previous systems have used largely pointwise properties like color at a pixel for initial hypotheses. ACRONYM addresses generic model classes and generic viewing, and extensive use of shape. Representation is central to ACRONYM, analytic representation of three-dimensional shape of objects and their two-dimensional appearances and representations of tasks and programs. ACRONYM takes a structural approach to interpretation. ACRONYM represents objects by segmenting them into part/whole graphs whose parts are volumes represented as **generalized cones**. Generalized cones are snakelike parts whose cross sections change smoothly along a space curve called the axis. Figure 8 shows a collection of generalized cone primitives from ACRONYM. Surfaces are represented as **ribbons**, which are specializations of generalized cones to surfaces. An aid to interpretation is a formalization of spatial matching.

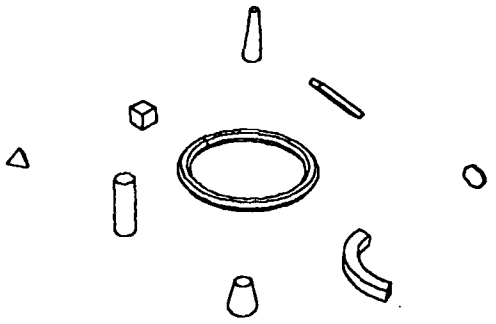


Figure 8. Generalized Cone Examples

Segmentation is a recurring problem at all levels in vision systems. An important limitation on image understanding systems is the "signal to symbol" transformation. A vigorous program is being pursued to develop image shape descriptors and surface shape descriptors. Both edge-based and stereo segmentation are being pursued.

The user inputs models to ACRONYM in a compact high level language with the aid of a model editor. In the predictive paradigm, ACRONYM determines symbolic observables of the appearance of objects from their models, particularly those that are quasi-invariant, approximately stable over a large range of objects and viewing classes. ACRONYM plans a sequence of observations to tentatively identify the object. ACRONYM then determines model parameters to match models with observed objects in order to make detailed verification.

Development of ACRONYM has stressed goal-directed interpretation (top-down) for problems like aircraft monitoring. Generalized cone representations of parts of objects embody powerful notions of local structure and well-formed objects. In the descriptive paradigm, ACRONYM will use quasi-invariants and generalized cone models to build up ribbon descriptions of images and generalized cone descriptions of spatial structure [Nevatia and Binford 77]. None of the mechanisms of that earlier work have yet been included in ACRONYM, however that work will be resumed. There is an intimate connection between predictive and descriptive parts in that description uses very general forms of prediction.

ACRONYM has four major parts: modeling system, Predictor and Planner, Interpreter, and image descriptor system.

Predictor and Planner

The Predictor and Planner uses geometric reasoning to predict the appearance of object parts and to make a plan (program) for interpretation. It works from a 3d model of an object and the context of the object (e.g. an aircraft is on the ground on an airfield), from which it derives generic and specific expectations for surface and image observables.

A prototype Predictor and Planner is complete. Sixty rules have been built for the first task, identification of aircraft, enough that ACRONYM identified an L1011 using the rule base. Most of these rules will be valuable for the next object class, vehicles. We expect that after the first four or five object classes the rule base will be nearly complete and well-structured, and that few additional rules will be necessary for other object classes.

The rule interpreter has been augmented to allow more compact and more powerful rules, and to incorporate the information contained in Object Graph and Observability Graph. We are extending the rule interpreter with a more general pattern directed invocation scheme. Explicit binding will make a cleaner and simpler structure for the person writing the rules and for meta-rules. Control of the reasoning process now includes forward reasoning with an agenda with preconditions on items. The rule base is being restructured.

The reasoning system is a production rule system with the addition of the Object Graph and the Observability Graph. It is consequent-driven (i.e. backward-chained); initially the

system was modeled after MYCIN [Davis 1975]. Much of ACRONYM's knowledge is maintained in the Object Graph and Observability Graph, not just in the rules. Thus, changes were made which make the Predictor and Planner heavily dependent on use of Object Graph. Changes to the rule interpreter allow a clean way to do within the rules what were previously side effects. The backward chaining strategy finds all rules which might satisfy the current goal. They are tried in turn until one succeeds. If all premises (antecedents) of a rule are satisfied the side effects are carried out and the consequents are asserted. The rule interpreter could only deal with the antecedents and consequents, not the side effects. Much of the action was outside the rule base; no reasoning with side effects could be included in writing meta-rules. The new scheme is essential for further development of geometric reasoning.

Rules can be categorized into classes of rules about the camera model, rules for deducing simple shapes, rules to calculate relative positions of objects, rules for symbolic arithmetic, rules that build Observability Graph nodes, and rules that transfer values from the Object Graph to the memory of assertions. There are rules to handle quantification. Other rules determine model parameters from image measurements. Thus far, rules deal only with simple generalized cones, flat straight ribbons (such as runways), and right circular cylinders.

Interpreter

The interpreter is a graph-matching procedure which uses relaxation. It is relatively well-developed, however minor problems were discovered in working through examples. Changes were made for more efficient matching by instantiating relations between nodes rather than single nodes. The Interpreter knows nothing about geometry, nothing about perception. It is now a domain-independent graph matcher. Progress has been made in redesign of the interpreter to introduce geometric knowledge and rule-based geometric reasoning, to divide the effort more between planning in advance and reasoning at the time of interpretation. In the future, the Interpreter will probably merge with the Predictor and Planner. This will be necessary in using rules for efficiency, and in integrating bottom-up descriptive mechanisms. Merging the Interpreter in the rule-based mechanism will make it easier to program.

We are formalizing interpretation in terms of transformational invariance, an extension of the principle of translational invariance which underlies the generalized cone representation.

The Interpreter performs a graph match by first matching nodes of the Observability Graph, then matching arcs. It uses relaxation to establish consistent assignments. The Interpreter is capable of doing much more than the Predictor and Planner can ask of it now, however it is very hard to program to use that generality. The Interpreter is really programmed by the Observability Graph. Control can be programmed into the Observability Graph. That form of control is not explicit; it is difficult for the Predictor and Planner to control it indirectly. Thus other forms of control are being considered. It is thought that dynamic planning will extend the power of the interpretation process.

Modeling System

The user communicates with the modeling system to build models of objects and their contexts. A high level modeling language has been built which reflects powerful underlying modeling primitives. Models are built of primitives which are generalized cones. Construction is simplified by having a library of prototype volumes and typical spines, cross sections, and sweeping rules from which to build generalized cones. The modeling system helps to construct models by modifying other models. Models are defined with a hierarchy of levels of detail; thus fine detail can be modeled at little cost.

ACRONYM requires models with generic

parts, generic and symbolic predictions of appearance. If ACRONYM were not concerned with interpretation, if it were only a graphics system, conventional modeling and graphics systems would be adequate.

Generalized cones are defined by a cross section, a space curve called the spine, and a sweeping rule which defines the transformation of the cross section along the spine. Cross sections may be made of straight lines and arcs of circles; spines may be made of straight lines and circular arcs; sweeping rules are constant or linear. The model parser accepts descriptions of cross sections, axes, and sweeping rules, with some cases symbolically defined. The class of generalized cones fully implemented includes: cross sections composed of straight lines and circular arcs with straight axes and linear sweeping rules; convex polygonal cross sections with axes made of straight lines and circular arcs, with linear sweeping rules.

New capabilities simplify the task of the user and expand ACRONYM's modeling primitives. An interactive geometric editor (MODITOR) has been built to assist the user and give him easy control over model building. MODITOR operates at the textual level. It aids the user to edit the Object Graph, filling and changing slots in generalized cone descriptors. Work is underway to make the geometric editor operate at the geometric level of manipulating objects, specifying translations and rotations, sweeping cross sections along curves, etc. There will be a third level, an intelligent editor which uses the rule-based system to manipulate relations between parts in a natural way.

We have implemented a new class of generalized cones in ACRONYM. In order to describe parts of the L1011 fuselage and especially wings, it was found useful to implement generalized cones with cross sections which are not perpendicular to the cone axis. It can be seen most clearly in aircraft wings. Wing cross sections are natural along streamlines, that is fore and aft, while the wing surface is continuous along the direction of the swept wing, which is not orthogonal to streamlines. We have made progress toward extending our mathematical analysis for representation to a powerful subclass of generalized cones which is complete in the sense of splines, generating surfaces which are made of pieces which are continuous with continuous tangent and curvature at joints.

The user gets feedback from the modeling system by a graphics display of models. The system has a line display which has back surface suppression but not full hidden surface suppression. Its solution for apparent edges (e.g. the edges of a cylinder which are not true edges) and back surface suppression is analytic and very fast. Work is underway toward a full hidden surface algorithm.

Stereo and Feature Descriptors

Effective surface description involves stereo mapping to produce a depth map, and surface segmentation to produce a symbolic description of surfaces as separate planes, cylinders, etc. Gennery has built a symbolic description system which takes the output of his stereo mapping system, determines the ground plane, segments objects above the ground plane, then describes those objects as ellipsoids or clusters of ellipsoids [Gennery 79].

Stereo

Baker has developed an edge-based stereo mapping system which searches for consistent edge pairings line-by-line along epipolar lines. It then rejects pairings which violate interline continuity. It determines edges by a very simple step of taking zero crossings of the "laterally inhibited" signal, the difference from the local average. Edges are obtained by zero crossings of 1x7 and 7x1 bar masks. Implementation of the system was motivated by a desire for speed; it executes in about 30 seconds for 256x256 images. Edges are paired on the basis of matching contrast or intensity on one side of the edge. For each edge there are a set of possible matches. Formerly, a branch and bound search process was used to

establish correspondence. That has been replaced by a Viterbi algorithm dynamic programming method. The Viterbi algorithm is much faster. It returns only a single best solution however, which may be error sensitive. That is, the locally optimal solution for single lines may not be globally consistent. A later stage removes some of these errors. The program uses a coarse to fine search procedure like the binary search correlator introduced by [Moravec 77], however limited to single epipolar lines. Branch and bound search and dynamic programming both sought solutions which maximize the number of edges which matched; among solutions with equal numbers of edge matches, they sought solutions which minimize the sum of squared errors of intensities of intervals to either side. Pairings for individual epipolar lines are tested for consistency by testing depth continuity. The consistency procedure follows connected edges in both images, calculating local mean and standard deviation of disparity, and removing pairings by a sequence of tests on disparity.

SAIL-IU is designing an advanced stereo mapping system which will produce measurements of surface boundaries accurate to optical and quantization limits (.12 pixels rms with reasonable contrast). The new stereo system is still in the design phase. It is based on a coarse-to-fine matching scheme first used by Moravec in 1974. That binary search strategy leads to efficiency but also eliminates ambiguities by including context of the area being matched. The new system will use both edge correspondence [Arnold 1978] and area correspondence [Moravec 1977, Gennery 1977].

It will use successive refinement, a different form of coarse-to-fine correspondence. Successive iterations concentrate on matching features and areas which have no correspondence. The new system will make surface interpretations with an improved measure for interpretation. It will have improved curved edges and make use of ribbon surfaces to describe curved surfaces.

We are making a theoretical analysis of consistency constraints which follow from obscuration and simplicity assumptions. We are also making statistical analysis of noise effects for line finding to maintain tight constraints. Frame to frame coherence is being incorporated for image sequences.

Edge Segmentation

Progress has been made in extending concepts of the Binford-Horn edge finding and curve linking program. Binford-Horn was based on several phases: 1. Detect edges using directional differences of the "laterally-inhibited" signal (subtraction of the local average, similar to difference of gaussians but with step functions). 2. Localize edges from zero crossing of the "laterally inhibited" signal; in essence, detection based on a large odd component, localization at the zero of the even component. 3. Reject or keep segments by tracing them with a sequential detection process.

An intermediate level edge-finding scheme is being implemented based on simplification of *Binford-Horn. This intermediate level system* is aimed toward VLSI implementation.

The Goal-Directed Ribbon Finder

Brooks has made a goal-directed module meant to be guided by ACRONYM [Brooks 1979b]. It is intended to be invoked many times with different heuristics and parameters, based on previous results from edge linking and interpretation. Thus, it is a family of goal-directed descriptor mechanisms. It uses line segments from [Nevatia and Babu 78]. An example is shown in Figure 3. An example of its ribbon output is shown in Figure 4.

There are four phases of the algorithm: 1. linking segments into candidate contours; 2. discarding unsatisfactory candidates; 3. finding ribbon descriptors for contours and the regions they bound; and 4. disambiguating redundant descriptions of a single region of the image.

Linking edges into a contour is formulated as a tree search. The linking phase selects starting

nodes for the search, e.g. restricting starting segments to a part of an image, or a certain direction. Edge segments are hashed by their image location; tests for proximity and image location are thus efficient. A best-first search is made from the starting node. Candidates for the tree search are generated by heuristics. Candidate nodes are pruned by another set of heuristics. Search is limited by expanding only a given number of nodes (typically 2). Heuristics for generating candidates are primarily smooth continuation and proximity. Pruning heuristics include a crude test for convexity.

Contours are linked by local properties. A set of heuristics screens these contours. Ribbon descriptors are fit to contours which survive. The program describes the class of ribbons bounded by two straight lines. Boundary directions are found from maxima of a histogram of edge directions. Two straight lines are fit to those boundary segments.

Often there will be multiple ribbons which overlap the same area. In some cases, several ribbons will be useful. In the example shown, the rear engine pod and the fuselage overlap. Both are useful. Disambiguation is done by the Interpreter.

About 30 seconds KL-10 CPU time was required for the example shown. The system has developmental baggage and has not been optimized.

ACRONYM performance

The total system for identification of the L1011 required about 110k 36 bit words. Some of this is necessary at the time of model-building, some at the time of planning, some at execution time. The modeler contains about 5k of parser code and 23k for graphics and tv buffer. The predictor and planner requires about 2k code, the matcher about 10k of code, the ribbon finder about 7k code. The LISP system requires about 48k with record package and i/o. Data structures require about 18k. A guess is that 200 rules will be required for aircraft monitoring and eventually 500 for a complete system. At an average of 30 list cells for each rule, 500 rules would require 15-30k for the rule base. A minimal execution system without planning is about 70k. Since then ACRONYM has grown considerably. The ribbon finder requires about 30 sec, matching about 1 sec,, the Predictor and Planner requires 1 second uncompiled, the modeler 1 second with file i/o of 1 second, and graphics of .5 seconds.

Passive Navigation

The Stanford stereo system is being used in the Navigation with Passive Sensors program [Firschein 1979]. It has obtained automatic registration and a camera model for pairs of images from the Night Vision Lab terrain model. Lockheed is evaluating the stereo system for two roles: first, terrain mapping of relative elevations for use in navigation by terrain matching (passive TERCOM); second, to measure V/H in a navigation system which flies by dead reckoning between landmarks. In the latter role, the stereo system can function on a non-stabilized platform. Baker has demonstrated good stereo reconstruction of terrain from the NVL terrain model. Terrain elevation data are not yet available from NVL to verify quantitatively the quality of the elevation reconstruction.

The other part of the passive navigation program is tracking of linear features. The work on extensions of Binford-Horn edge-finding and curve-linking is of key importance here, however the work is not yet complete.

5.4 Autonomous Navigation

Hans Moravec recently completed a dissertation on visual navigation based on experiments with a robot vehicle. The Stanford AI lab cart is a card-table sized mobile robot controlled remotely through a radio link, and equipped with a TV camera and transmitter. A computer has been programmed to drive the cart through cluttered indoor and outdoor spaces, gaining its knowledge about the world entirely from images broadcast by the onboard TV system.

The cart deduces the three dimensional location of objects around it, and its own motion among them, by noting their apparent relative shifts in successive images obtained from the moving TV camera. It maintains a model of the location of the ground, and registers objects it has seen as potential obstacles if they are sufficiently above the surface, but not too high. It plans a path to a user-specified destination which avoids these obstructions. This plan is changed as the moving cart perceives new obstacles on its journey.

The system is moderately reliable, but very slow. The cart moves about one meter every ten to fifteen minutes, in lurches. After rolling a meter, it stops, takes some pictures and thinks about them for a long time. Then it plans a new path, and executes a little of it, and pauses again.

The program has successfully driven the cart through several 20 meter indoor courses (each taking about five hours) complex enough to necessitate three or four avoiding swerves. A less successful outdoor run, in which the cart swerved around two obstacles but collided with a third, was also done. Harsh lighting (very bright surfaces next to very dark shadows) resulting in poor pictures, and movement of shadows during the cart's creeping progress, were major reasons for the poorer outdoor performance. These obstacle runs have been filmed (minus the very dull pauses).

A typical run of the avoider system begins with a calibration of the cart's camera. The cart is parked in a standard position in front of a wall of spots. A calibration program notes the disparity in position of the spots in the image seen by the camera with their position predicted from an idealized model of the situation. It calculates a distortion correction polynomial which relates these positions, and which is used in subsequent ranging calculations.

The cart is then manually driven to its obstacle course. Typically this is either in the large room in which it lives, or a stretch of the driveway which encircles the AI lab. Chairs, boxes, cardboard constructions and assorted debris serve as obstacles in the room. Outdoors the course contains curbing, trees, parked cars and signposts as well.

The obstacle avoiding program is started. It begins by asking for the cart's destination, relative to its current position and heading. After being told, say, 50 meters forward and 20 to the right, it begins its maneuvers.

It activates a mechanism which moves the TV camera, and digitizes about nine pictures as the camera slides (in precise steps) from one side to the other along a 50 cm track.

A subroutine called the interest operator described in is applied to the one of these pictures. It picks out 30 or so particularly distinctive regions (features) in this picture. Another routine called the correlator looks for these same regions in the other frames. A program called the camera solver determines the three dimensional position of the features with respect to the cart from their apparent movement image to image.

The navigator plans a path to the destination which avoids all the perceived features by a large safety margin. The program then sends steering and drive commands to the cart to move it about a meter along the planned path. The cart's response to such commands is not very precise.

After the step forward the camera is operated as before, and nine new images are acquired. The control program uses a version of the correlator to find as many of the features from the previous location as possible in the new pictures, and applies the camera solver. The program then deduces the cart's actual motion during the step from the apparent three dimensional shift of these features.

The motion of the cart as a whole is larger and less constrained than the precise slide of the camera. The images between steps forward can vary greatly, and the correlator is usually

unable to find many of the features it wants. The interest operator/correlator/camera solver combination is used to find new features to replace lost ones.

The three dimensional location of any new features found is added to the program's model of the world. The navigator is invoked to generate a new path that avoids all known features, and the cart is commanded to take another step forward.

This continues until the cart arrives at its destination or until some disaster terminates the program.

Previous Results: Stereo

Research has demonstrated high potential for utility of passive imaging techniques for high resolution depth measurement. Passive techniques have important advantages over active ranging techniques in hostile environments. Sequences of images from a moving aircraft have been used to find the ground plane and separate objects from ground. The system should be effective with camouflaged surfaces. Accuracy of 2" height error has been attained for 3" horizontal pixel size on the ground, with a 60 degree baseline. On a general purpose computer, the process requires about 15 seconds with no guidance information. That can likely be reduced at least a factor of 2. With accurate guidance information, the time required is estimated to be about 250 msec (most missions would probably be in this category). The system is self-calibrating and reliable.

Automatic Registration

The system includes a solution to the problem of determining the stereo camera model from information within the pair of pictures. Imagine an aircraft approaching a runway. As it moves, objects on both sides appear to move radially outward from a center, the fixed point. The fixed point is the instantaneous direction of motion. The pilot knows that the point which does not appear to move is where he will touch down, unless he changes direction. The distance between views and the apparent displacement of points allow calculation of the distance of each point from the observer and from the vehicle path. The touchdown point can be calculated from the trajectory of centers. That is precisely what is done by the camera transform solver in the system [Gennery 1977]. It determines the transform from one view to another in a sequence of views from a moving observer. The program first orients itself in the scene and finds a model for the transform between the two cameras. If two views are an accurately calibrated stereo pair, this operation is not necessary. If accurate guidance information is available, this operation can be speeded up enormously. The program finds a camera transform model by finding a sample of features of interest in one image and matching them with their corresponding view in the other image. The Interest Operator requires about 75 msec for a 256x256 frame. Interesting features are areas (typically 8x8) which can be localized in two dimensions without a camera transform. The operator chooses those areas with large variance along all grid directions. That is roughly equivalent to a large drop in autocorrelation along all grid directions, which means that the area can be localized closely.

The binary search correlator matches the features in the other image by a coarse to fine strategy: It has versions of the picture at resolutions of 256x256, 128x128, 64x64, 32x32, and 16x16. It first matches by a small search on a very coarse version (16x16) of the image. It then performs a search in the next finer version of the image, in the neighborhood of the best match in the previous image. That step is repeated until the full resolution is reached. The matching process requires only 50msec for a match of a single feature no matter where it is in the image. If the camera transform is known, search is necessary only along a line in the image. In this case, search is about a factor of seven faster. If the depth of neighboring points is used as a starting point for the search, the match is another factor of seven faster. It is planned to incorporate those speedups; neither is now used.

The matching has about 10% errors. It encounters fewer ambiguities than brute force matching, since not only must the feature match, but the surrounding context must also match. The procedure should not work for parts of scenes where the background of objects (context) changes drastically from one view to another. This is true only at very wide angles and close range. Aerial views are mostly planar, so failure of matching should not be a problem, nor has it been in practice. The process requires about 50k of 36 bit words now. It is possible to implement the coarse-to-fine search strategy in a raster scan and keep only a portion of each image in core. This would cut memory size by a large amount, but it has not been done.

Given corresponding views of at least five points which are non-degenerate (i.e. no colinear and planar degeneracies) the relative transform of the two views can be found. It is not necessary to know the position of these points, only which views correspond. The transform is determined except for a scale factor, the length of the stereo baseline. That does not affect subsequent matching of two views using the transform, and the scale factor can often be determined from known scene distances or guidance information. If the scene is nearly flat, then certain parameters are ill-determined. However, that does not affect the accuracy of measuring heights using the transform. If the scene is nearly flat, then a special simplified form of the solver can be used.

The special case version has been used on some images. It is much faster than the full transform solver. Part of the job of the transform solver is to deal with mistaken matches. The procedure calculates an error matrix for each point and iterates by throwing out wild points. It calculates an error matrix from which errors in depths of point pairs are calculated. The solver uses typically 12 points and requires about 300 msec per point. It requires about 20k of memory. With accurate guidance information, this operation would not be necessary. However, it can be used directly to find the instantaneous direction of the vehicle. As mentioned above, as the vehicle moves, points in the image appear to move radially away from the center which is the instantaneous direction of the vehicle. Three angles relate the coordinate system of one view with the other, and two angles specify the direction of the instantaneous direction of motion.

The stereo system was used to register a pair of pictures of the Pittsburgh skyline. Initially, the solution failed to converge; when several erroneous matches were excluded, the solution converged well.

Ground Plane

The camera transform model makes it economical to make a denser depth map. A point in one view corresponds to a ray in space which corresponds to a line in the other view. The search is limited to this line, and in addition, nearby points usually have about the same disparity as their neighbors. Thus, search is limited to a small interval on a line. A high resolution correlator has been developed which interpolates to the best match, and which calculates the precision of the match based on statistics of the area.

The system then finds a best ground plane or ground parabola to the depth points, in the least squares sense. It gives no weight to points above the ground plane; it expects many of those. It includes points below the ground plane and near the ground plane. Since points below the ground plane may be wild points, they are edited out in an iterative procedure. Of course, there may be holes. If they are small, there is no problem. If the hole is big, it becomes the ground plane. The ground plane finder requires 5 msec per point.

Edge-Based Stereo

Feature-based stereo using edges increases the accuracy with which boundaries of depth discontinuities can be found by about a factor of 25. It also provides additional information

about surface markings which are not available in stereo based on area correlation. Feature-based stereo is potentially fast, although for the moment, the binary search correlator is faster. Edge-based techniques have not been developed very far, and would benefit from "smart sensor" technology. A new technique has been developed to use edge features in stereo. Edges are linked along smooth curves in 3d (in the image coordinates and in depth). The new technique is used in the object modeling and recognition modules of the system. Those edges out of the ground plane delimit bodies, if isolated.

In a stereo pair of images of an aircraft at San Francisco airport, the succession of edges matched in stereo as a function of height showed a separation in height between wing tips and and wing roots of an L1011.

Vehicle Location

Arnold [1977] demonstrated progress toward vehicle location and identification. The system registered images of a suburban parking lot and obtained the stereo camera model. It separated the vehicles from the ground and succeeded in describing the projection of a car by a rectangle of approximately the right size and orientation. The length and width of the car were accurate to about 5% by inspection.

Recognition of Complex Objects

Nevatia developed a system which took depth maps of a doll, a toy horse, a glove, a hammer and a ring using depth maps from a laser triangulation system [Nevatia 1974]. These depth maps were described in terms of generalized cones which were organized into part/whole structures. The system recognized different views of these objects with articulation of limbs and some obscuration. The system addressed important issues of representation: indexing into a subclass of similar objects in visual memory instead of matching against all stored models; determining generalized cone descriptors from surface descriptors.

5.5 References

[Arnold 1977] Arnold,R.D.; *"Spatial Understanding"*; Proc Image Understanding Workshop, April 1977.

[Arnold 1978] Arnold,R.D.; *"Local Context in Matching Edges for Stereo Vision"*; Proc Image Understanding Workshop, Boston, May 1978.

[Barrow 1976] Barrow,H.G., Tenenbaum, J.M.; *"MSYS: A System for Reasoning about Scenes"*; SRI AI Center Tech Note 121, April 1976.

[Binford 1979] Binford, T.O., Brooks, R.A.; *"Geometric Reasoning in ACRONYM"*; Proc Image Understanding Workshop", Palo Alto, Calif, Apr 1979.

[Brooks 1978a] Brooks, R.A., Greiner, R., Binford, T.O.; *"A Model-Based Vision System"*; Proceedings Image Understanding Workshop", Boston, May 1978.

[Brooks 1978b] Brooks, R.A., Greiner, R., Binford, T.O.; *"Progress Report on a Model-Based Vision System"*; Proceedings Image Understanding Workshop", Carnegie-Mellon, Nov 1978.

[Brooks 1979b] Brooks, R.A.; *"Goal-Directed Edge Linking and Ribbon Finding"*; Proc Image Understanding Workshop, Palo Alto, Calif, Apr 1979.

[Brooks 79c] Brooks, R.A., Greiner, R., Binford, T.O.; *"ACRONYM: A Model-Based Vision System"*; Proc Int Jt Conf on AI, Aug 1979.

[Davis 1975] Davis, R., Buchanan, B., Shortliffe, E.; *"Production Rules as a Representation for a Knowledge-Based Consultation Program"*; Stanford A I Memo AIM-266, 1975.

[Firschein 1979] [Firschein, O., Gennery, D. Milgram, D. Pearson, J.J.; *"Progress in Navigation Using Passively Sensed Images"*; Proc ARPA Image Understanding Workshop, Palo Alto, 1979.

[Gennery 1976] Gennery, D.B.; *"Stereo Camera Solver"*; Stanford A I Lab Internal Note, 1976.

[Gennery 1977] Gennery, D.B.; *"A Stereo Vision System"*; Proc Image Understanding Workshop, October 1977.

[Gennery 1977a] Gennery, D.B.; *"Ground Plane Finder"*; Stanford A I Lab Internal Note, 1977.

[Gennery 1979] Gennery, D.B.; *"Object Detection and Measurement using Stereo Vision"*; Draft Feb 1979 submitted to International Joint Conf on AI, Aug 1979.

[Hannah] Hannah, M.J.; *"Computer Matching of Areas in Stereo Images"* Artificial Intelligence Laboratory, Stanford University, Memo AIM-239 July 1974.

[Horn 72] B.K.P.Horn; *"The Binford-Horn Edge Finder"*; MIT AI Memo 285, revised December 1973.

[Kanade 1977] Kanade, T.; *"Model Representations and Control Structures in Image Understanding"*; Proc IJCAI-5, Boston, 1977.

[Marr and Poggio] D. Marr and T. Poggio; *"A Theory of Human Stereo Vision,"* AI Memo 451, MIT, Nov 1977.

[Moravec 1977] Moravec, H.P.; *"Towards Automatic Visual Obstacle Avoidance"*; Proc 5th International Joint Conf on Artificial Intelligence, MIT, Boston, Aug 1977.

[Moravec 1979] Moravec, H.P.; *"Visual Mapping by a Robot Rover"*, Draft Feb 1979 submitted to Int Joint Conf on AI, Aug 1979.

[Nevatia 1974] Nevatia, R.K.; *"Structured Descriptions of Complex Curved Objects for Recognition and Visual Memory"*; Stanford A I Lab Memo AIM-250.

[Nevatia 76] Nevatia, R.K.; *"Depth Measurement by Motion Stereo"*; Comp. Graph. and Image Proc., vol.5 , 1976, pp. 203-214.

[Nevatia and Binford 1977] Nevatia, R.K., Binford, T.O.; *"Description and Recognition of Curved Objects"*; Artificial Intelligence 9, 77, 1977.

[Nevatia and Babu 1978] Nevatia, R.K., Babu, K.R.; *"Linear Feature Extraction"*; Proc ARPA Image Understanding Workshop, CMU, Nov 1978.

[Panton 1978] Panton, D.J.; *"A Flexible Approach to Digital Stereo Mapping"*; (CDC) Photogrammetric Engineering and Remote Sensing V 44, p1499, Dec 1978.

[Rubin 1978] Rubin, S.M.; *"The ARGOS Image Understanding System"*; Proc ARPA Image Understanding Workshop, CMU, Nov 1978.

[Tenenbaum 1976] Tenenbaum, J.M., Barrow, H.G.; *"Experiments in Interpretation Guided Segmentation"*; SRI AI Center Tech Note 123.

[Waltz 1972] Waltz,D.; *"Generating Semantic Descriptions from Drawings of Scenes with Shadows"*; MIT-AI Tech Rept AI-TR-271, 1972.also *"Understanding Line Drawings of Scenes with Shadows"*; In "The Psychology of Computer Perception"; P.H.Winston, ed; McGraw-Hill, 1975.

Appendix A
Theses

Theses that have been published by this laboratory are listed here. Several earned degrees at institutions other than Stanford, as noted. This list is kept in diskfile THESES [BIB,DOC] @SU-AI.

D. Raj. Reddy, AIM-43
An Approach to Computer Speech Recognition by Direct Analysis of the Speech Wave,
Ph.D. in Computer Science,
September 1966.

S. Persson, AIM-46
Some Sequence Extrapolating Programs: a Study of Representation and Modeling in Inquiring Systems,
Ph.D. in Computer Science, University of California, Berkeley,
September 1966.

Bruce Buchanan, AIM-47
Logics of Scientific Discovery,
Ph.D. in Philosophy, University of California, Berkeley,
December 1966.

James Painter, AIM-44
Semantic Correctness of a Compiler for an Algol-like Language,
Ph.D. in Computer Science,
March 1967.

William Wichman, AIM-56
Use of Optical Feedback in the Computer Control of an Arm,
Eng. in Electrical Engineering,
August 1967.

Monte Callero, AIM-58
An Adaptive Command and Control System Utilizing Heuristic Learning Processes,
Ph.D. in Operations Research,
December 1967.

Donald Kaplan, AIM-60
The Formal Theoretic Analysis of Strong Equivalence for Elemental Properties,
Ph.D. in Computer Science,
July 1968.

Barbara Huberman, AIM-65
A Program to Play Chess End Games,
Ph.D. in Computer Science,
August 1968.

Donald Pieper, AIM-72
The Kinematics of Manipulators under Computer Control,
Ph.D. in Mechanical Engineering,
October 1968.

Donald Waterman, AIM-74
Machine Learning of Heuristics,
Ph.D. in Computer Science,
December 1968.

Roger Schank, AIM-83
A Conceptual Dependency Representation for a Computer Oriented Semantics,
Ph.D. in Linguistics, University of Texas,
March 1969.

Pierre Vicens, AIM-85
Aspects of Speech Recognition by Computer,
Ph.D. in Computer Science,
March 1969.

Victor D. Scheinman, AIM-92
Design of Computer Controlled Manipulator,
Eng. in Mechanical Engineering,
June 1969.

Claude Cordell Green, AIM-96
The Application of Theorem Proving to Question-answering Systems,
Ph.D. in Electrical Engineering,
August 1969.

James J. Horning, AIM-98
A Study of Grammatical Inference,
Ph.D. in Computer Science,
August 1969.

Michael E. Kahn, AIM-106
The Near-minimum-time Control of Open-loop Articulated Kinematic Chains,
Ph.D. in Mechanical Engineering,
December 1969.

Joseph Becker, AIM-119
An Information-processing Model of Intermediate-Level Cognition,
Ph.D. in Computer Science,
May 1972.

Irwin Sobel, AIM-121
Camera Models and Machine Perception,
Ph.D. in Electrical Engineering,
May 1970.

Michael D. Kelly, AIM-130
Visual Identification of People by Computer,
Ph.D. in Computer Science,
July 1970.

Gilbert Falk, AIM-132
Computer Interpretation of Imperfect Line Data as a Three-dimensional Scene,
Ph.D. in Electrical Engineering,
August 1970.

Jay Martin Tenenbaum, AIM-134
Accommodation in Computer Vision,
Ph.D. in Electrical Engineering,
September 1970.

Lynn H. Quam, AIM-144
Computer Comparison of Pictures,
Ph.D. in Computer Science,
May 1971.

Robert E. Kling, AIM-147
Reasoning by Analogy with Applications to Heuristic Problem Solving: a Case Study,
Ph.D. in Computer Science,
August 1971.

Rodney Albert Schmidt Jr., AIM-149
A Study of the Real-time Control of a Computer-driven Vehicle,
Ph.D. in Electrical Engineering,
August 1971.

Jonathan Leonard Ryder, AIM-155
Heuristic Analysis of Large Trees as Generated in the Game of Go,
Ph.D. in Computer Science,
December 1971.

Jean M. Cadiou, AIM-163
Recursive Definitions of Partial Functions and their Computations,
Ph.D. in Computer Science,
April 1972.

Gerald Jacob Agin, AIM-173
Representation and Description of Curved Objects,
Ph.D. in Computer Science,
October 1972.

Francis Lockwood Morris, AIM-174
Correctness of Translations of Programming Languages – an Algebraic Approach,
Ph.D. in Computer Science,
August 1972.

Richard Paul, AIM-177
Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm,
Ph.D. in Computer Science,
November 1972.

Aharon Gill, AIM-178
Visual Feedback and Related Problems in Computer Controlled Hand Eye Coordination,
Ph.D. in Electrical Engineering,
October 1972.

Ruzena Bajcsy, AIM-180
Computer Identification of Textured Visiual Scenes,
Ph.D. in Computer Science,
October 1972.

Ashok Chandra, AIM-188
On the Properties and Applications of Programming Schemas,
Ph.D. in Computer Science,
March 1973.

Gunnar Rutger Grape, AIM-201
Model Based (Intermediate Level) Computer Vision,
Ph.D. in Computer Science,
May 1973.

Yoram Yakimovsky, AIM-209
Scene Analysis Using a Semantic Base for Region Growing,
Ph.D. in Computer Science,
July 1973.

Jean E. Vuillemin, AIM-218
Proof Techniques for Recursive Programs,
Ph.D. in Computer Science,
October 1973.

Daniel C. Swinehart, AIM-230
COPILOT: A Multiple Process Approach to Interactive Programming Systems,
Ph.D. in Computer Science,
May 1974.

James Gips, AIM-231
Shape Grammars and their Uses
Ph.D. in Computer Science,
May 1974.

Charles J. Rieger III, AIM-233
Conceptual Memory: A Theory and Computer Program for Processing the Meaning Content of Natural Language Utterances,
Ph.D. in Computer Science,
June 1974.

Christopher K. Riesbeck, AIM-238
Computational Understanding: Analysis of Sentences and Context,
Ph.D. in Computer Science,
June 1974.

Marsha Jo Hannah, AIM-239
Computer Matching of Areas in Stereo Images,
Ph.D. in Computer Science,
July 1974.

James R. Low, AIM-242
Automatic Coding: Choice of Data Structures,
Ph.D. in Computer Science,
August 1974.

Jack Buchanan, AIM-245
A Study in Automatic Programming
Ph.D. in Computer Science,
May 1974.

Neil Goldman, AIM-247
Computer Generation of Natural Language From a Deep Conceptual Base
Ph.D. in Computer Science,
January 1974.

Bruce Baumgart, AIM-249
Geometric Modeling for Computer Vision
Ph.D. in Computer Science,
October 1974.

Ramakant Nevatia, AIM-250
Structured Descriptions of Complex Curved Objects for Recognition and Visual Memory
Ph.D. in Electrical Engineering,
October 1974.

Edward H. Shortliffe, AIM-251
MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection
Ph.D. in Medical information Sciences,
October 1974.

Malcolm C. Newey, AIM-257
Formal Semantics of LISP With Applications to Program Correctness
Ph.D. in Computer Science,
January 1975.

Hanan Samet, AIM-259
Automatically Proving the Correctness of Translations Involving Optimized Coded
PhD in Computer Science,
Ma[illegible]75.

David Canfield Smith, AIM-260
PYGMALION: A Creative Programming Environment
PhD in Computer Science,
June 1975.

Sundaram Ganapathy, AIM-272
Reconstruction of Scenes Containing Polyhedra From Stereo Pair of Views
Ph.D. in Computer Science,
December 1975.

Linda Gail Hemphill, AIM-273
A Conceptual Approach to Automated Language Understanding and Belief Structures: with Disambiguation of the Word For
Ph.D. in Linguistics,
May 1975.

Norihsa Suzuki, AIM-279
Automatic Verification of Programs with Complex Data Structures
Ph.D. in Computer Science,
February 1976.

Russell Taylor, AIM-282
Synthesis of Manipulator Control Programs From Task-Level Specifications
PhD in Computer Science,
July 1976.

Randall Davis, AIM-283
Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases
Ph.D. in Computer Science,
July 1976.

Rafael Finkel, AIM-284
Constructing and Debugging Manipulator Programs
Ph.D. in Computer Science,
August 1976.

Douglas Lenat, AIM-286
AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search
Ph.D. in Computer Science,
July 1976.

Michael Roderick, AIM-287
Discrete Control of a Robot Arm
Engineer in Electrical Engineering,
August 1976.

Robert C. Bolles, AIM-295
Verification Vision Within a Programmable Assembly System
Ph.D. in Computer Science,
December 1976.

Robert Cartwright, AIM-296
Practical Formal Semantic Definition and Verification Systems
Ph.D. in Computer Science,
December 1976.

Todd Wagner, AIM-304
Hardware Verification
PhD in Computer Science,
September 1977.

William Faught, AIM-305
Motivation and Intensionality in a Computer Simulation Model
Ph.D. in Computer Science,
September 1977.

David Barstow, AIM-308
Automatic Construction of Algorithms
Ph.D. in Computer Science,
December 1977.

Bruce E. Shimano, AIM-313
The Kinematic Design and Force Control of Computer Controlled Manipulators
Ph.D. in Mechanical Engineering,
March 1978.

Jerrold M. Ginsparg, AIM-316
Natural Language Processing in an Automatic Programming Domain
Ph.D. in Computer Science,
June 1978.

Robert Elliot Filman, AIM-327
The Interaction of Observation and Inference
PhD in Computer Science,
April 1979.

David Edward Wilkins, AIM-329
Using Patterns and Plans to Solve Problems and Control Search
PhD in Computer Science,
June 1979.

Elaine Kant, AIM-331
Efficiency Consideration in Program Synthesis: A Knowedge-Based Approach
PhD in Computer Science,
July 1979.

Brian P. McCune, AIM-333
Building Program Models Incrementally from Informal Descriptions
PhD in Computer Science,
October 1979.

Appendix B
Film Reports

Prints of the following films are available for distribution. This list is kept in diskfile FILMS [BIB,DOC] @SU-AI.

1. Art Eisenson and Gary Feldman, **Ellis D. Kroptechev and Zeus, his Marvelous Time-sharing System**, 16mm B&W with sound, 15 minutes, March 1967.

The advantages of time-sharing over standard batch processing are revealed through the good offices of the Zeus time-sharing system on a PDP-1 computer. Our hero, Ellis, is saved from a fate worse than death. Recommended for mature audiences only.

2. Gary Feldman, **Butterfinger**, 16mm color with sound, 8 minutes, March 1968.

Describes the state of the hand-eye system at the Artificial Intelligence Project in the fall of 1967. The PDP-6 computer getting visual information from a television camera and controlling an electrical-mechanical arm solves simple tasks involving stacking blocks. The techniques of recognizing the blocks and their positions as well as controlling the arm are briefly presented. Rated "G".

3. Raj Reddy, Dave Espar and Art Eisenson, **Hear Here**, 16mm color with sound, 15 minutes, March 1969.

Describes the state of the speech recognition project as of Spring, 1969. A discussion of the problems of speech recognition is followed by two real time demonstrations of the current system. The first shows the computer learning to recognize phrases and second shows how the hand-eye system may be controlled by voice commands. Commands as complicated as 'Pick up the small block in the lower lefthand corner', are recognized and the tasks are carried out by the computer controlled arm.

4. Gary Feldman and Donald Peiper, **Avoid**, 16mm color, silent, 5 minutes, March 1969.

An illustration of Peiper's Ph.D. thesis. The problem is to move the computer controlled mechanical arm through a space filled with one or more known obstacles. The film shows the arm as it moving through various cluttered environments with fairly good success.

5. Richard Paul and Karl Pingle, **Instant Insanity**, 16mm color, silent, 6 minutes, August, 1971.

Shows the hand/eye system solving the puzzle *Instant Insanity*. Sequences include finding and recognizing cubes, color recognition and object manipulation. [Made to accompany a paper presented at the 1971 IJCAI. May be hard to understand without a narrator.]

6. Suzanne Kandra, **Motion and Vision**, 16mm color, sound, 22 minutes, November 1972.

A technical presentation of three research projects completed in 1972: advanced arm control by R. P. Paul [AIM-177], visual feedback control by A. Gill [AIM-178], and representation and description of curved objects by G. Agin [AIM-173]. Drags a bit.

7. Larry Ward, **Computer Interactive Picture Processing**, (MARS Project), 16mm color, sound, 8 min., Fall 1972.

This film describes an automated picture differencing technique for analyzing the variable surface features on Mars using data returned by the Mariner 9 spacecraft. The system uses a time-shared, terminal oriented PDP-10 computer. The film proceeds at a breathless pace. Don't blink, or you will miss an entire scene.

8. D.I. Okhotsimsky, et al, **Display Simulations of 6-Legged Walking**, Institute of Applied Mathematics – USSR Academy of Science, (titles translated by Stanford AI Lab and edited by Suzanne Kandra), 16mm black and white, silent, 10 minutes, 1972.

A display simulation of a 6-legged ant-like walker getting over various obstacles. The research is aimed at a planetary rover that would get around by walking. This cartoon favorite beats Mickey Mouse hands down. Or rather, feet down.

9. Richard Paul, Karl Pingle, and Bob Bolles, **Automated Pump Assembly**, 16mm color, silent (runs at sound speed!), 7 minutes, April, 1973.

Shows the hand-eye system assembling a simple pump, using vision to locate the pump body and to check for errors. The parts are assembled and screws inserted, using some special tools designed for the arm. Some titles are included to help explain the film.

10. Terry Winograd, **Dialog with a robot**, MIT A. I. Lab., 16mm black and white, silent, 20 minutes, 1971.

Presents a natural language dialog with a simulated robot block-manipulation system. The dialog is substantially the same as that in *Understanding Natural Language* (T. Winograd, Academic Press, 1972). No explanatory or narrative material is on the film.

11. Karl Pingle, Lou Paul, and Bob Bolles, **Programmable Assembly, Three Short Examples**, 16mm color, sound, 8 minutes, October 1974.

The first segment demonstrates the arm's ability to dynamically adjust for position and orientation changes. The task is to mount a bearing and seal on a crankshaft. Next, the arm is shown changing tools and recovering from a run-time error. Finally, a cinematic first: *two* arms cooperating to assemble a hinge.

12. Brian Harvey, **Display Terminals at Stanford**, 16mm B&W, sound, 13 minutes, May 1975.

Although there are many effective programs to use display terminals for special graphics applications, very few general purpose timesharing systems provide good support for using display terminals in normal text display applications. This film shows a session using the display system at the Stanford AI Lab, explaining how the display support features in the Stanford monitor enhance the user's control over his job and facilitate the writing of display-effective user programs.

13. M. Shahid Mujtaba, **POINTY - An Interactive System for Assembly**, 16mm color, sound, 10 minutes, December 1977.

POINTY is an interactive programming system that uses a mechanical manipulator as a measuring tool to determine the position and orientation of various parts laid out in a work station. Positions may be determined precisely by means of a sharp pointed tool held in the manipulator hand, or by using the finger touch sensors and moving the arm to the desired points either manually or under computer control. Arbitrary orientations may be determined from the location of three points. The data generated may be referred to symbolically, so that the programmer is freed from having to think in terms of the numerical values of object locations. The data is saved in a computer file for later use in a program to assemble the parts.

This film illustrates the use of POINTY instructions to collect the position data of two parts of a water valve assembly. It shows the use of multiple points to determine orientations, the procedure followed to obtain the data, and how the programmer may refer to the data symbolically. Finally, the arm is shown putting together the water valve assembly.

Appendix C
External Publications

Articles and books by project members that have appeared since July 1973 are listed here alphabetically by lead author. Earlier publications are given in our ten-year report [Memo AIM-228] and in diskfile PUBS.OLD [BIB,DOC] @SU-AI. The list below is kept in PUBS [BIB,DOC] @SU-AI.

1. Agin, Gerald J., Thomas O. Binford, **Computer Description of Curved Objects**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

2. Agin, G.J., T.O. Binford; **Representation and Description of Curved Objects**, *IEEE Transactions on Computers*, Vol C-25, 440, April 1976.

3. Aiello, L., Aiello, M. and Weyhrauch, R.W., **Pascal in LCF: Semantics and Examples of Proofs**, *Theoretical Computer Science*, 5, 1977.

4. Aiello, Luigia and G. Prini, **Design of a personal computing system**, *Proc. Annual Conf. of the Canadian Information Processing Society*, Victoria (British Columbia), May 12-14, 1980.

5. Aiello, Luigia and R. Weyhrauch, **Using meta-theoretic Reasoning to do Algebra**, *Proc. Automatic Deduction Conference*, Les Arcs (France), July 8-11, 1980.

6. Aiello, Mario, Richard Weyhrauch, **Checking Proofs in the Metamathematics of First Order Logic**, *Adv. Papers of 4th Int. Joint Conference on Artificial Intelligence*, Vol. 1, pp. 1-8, September 1975.

7. Arnold, R.D., **Local Context in Matching Edges for Stereo Vision**, *Proc. Image Understanding Workshop*, Boston, May 1978.

8. Ashcroft, Edward, Zohar Manna, Amir Pnueli, **Decidable Properties of Monodic Functional Schemas**, *J. ACM*, Vol. 20, No. 3, pp. 489-499, July 1973.

9. Ashcroft, Edward, Zohar Manna, **Translating Program Schemas to While-schemas**, *SIAM Journal on Computing*, Vol. 4, No. 2, pp. 125-146, June 1975.

10. Bajcsy, Ruzena, **Computer Description of Textured Scenes**, *Proc. Third Int. Joint Conf. on Artificial Intelligence*, Stanford U., 1973.

11. Barstow, David, Elaine Kant, **Observations on the Ineraction between Coding and Efficiency Knowledge in the PSI Program Synthesis System**, *Proc. 2nd Int. Conf. on Software Engineering*, IEEE Computer Society, Long Beach, California, October 1976.

12. Barstow, David, **A Knowledge-Based System for Automatic Program Construction**, *Proc. Int. Joint Conf. on A.I.*, August 1977.

13. Biermann, A. W., R.I. Baum, F.E. Petry, **Speeding Up the Synthesis of Programs from Traces**, *IEEE Trans. Computers*, February 1975.

14. Bobrow, Daniel, Terry Winograd, **An Overview of KRL, a Knowledge Representation Language**, *J. Cognitive Science*, Vol. 1, No. 1, 1977.

15. Bobrow, Dan, Terry Winograd, & KRL Research Group, **Experience with KRL-0: One Cycle of a Knowledge Representation Language**, *Proc. Int. Joint Conf. on A.I.*, August 1977.

16. Bolles, Robert C. **Verification Vision for Programmable Assembly**, *Proc. Int. Joint Conf. on A.I.*, August 1977.

17. Brooks, R., R. Greiner, and T.O. Binford, **A Model-Based Vision System**; *Proc. Image Understanding Workshop*, Boston, May 1978.

18. Cartwright, Robert S., Derek C. Oppen, **Unrestricted Procedure Calls in Hoare's Logic**, *Proc. Fifth ACM Symposium on Principles of Programming Languages*, January 1978.

19. Cartwright, Robert and John McCarthy, **First Order Programming Logic**, *Proc. 6th Annual ACM Symp. on Principles of Programming Languages*, San Antonio, Texas, January 1979.

20. Chandra, Ashok, Zohar Manna, **On the Power of Programming Features**, *Computer Languages*, Vol. 1, No. 3, pp. 219-232, September 1975.

21. Chowning, John M., **The Synthesis of Complex Audio Spectra by means of Frequency Modulation**, *J. Audio Engineering Society*, September 1973.

22. Clark, Douglas, and Green, C. Cordell, **An Empirical Study of List Structure in LISP**, *Communications of the ACM*, Volume 20, Number 2, February 1977.

23. Colby, Kenneth M., *Artificial Paranoia: A Computer Simulation of the Paranoid Mode*, Pergamon Press, N.Y., 1974.

24. Colby, K.M. and Parkison, R.C. **Pattern-matching rules for the Recognition of Natural Language Dialogue Expressions**, *American Journal of Computational Linguistics*, 1, September 1974.

25. Creary, Lewis G., **Propositional Attitudes: Fregean Representation and Simulative Reasoning**, *Proc. 6th Int. Joint Conference on Artificial Intelligence*, Tokyo, Japan, August 1979.

26. Dershowitz, Nachum, Zohar Manna, **On Automating Structural Programming**, *Colloques IRIA on Proving and Improving Programs*, Arc-et-Senans, France, pp. 167-193, July 1975.

27. Dershowitz, Nachum, Zohar Manna, **The Evolution of Programs: a system for automatic program modification**, *IEEE Trans. Software Eng.*, Vol. 3, No. 5, pp. 377-385, November 1977.

28. Dershowitz, Nachum, Zohar Manna, **Inference Rules for Program Annotation**, *Proc. 3rd Int. Conf. on Software Engineering*, Atlanta, Ga., pp. 158-167, May 1978.

29. Dobrotin, Boris M., Victor D. Scheinman, **Design of a Computer Controlled Manipulator for Robot Research**, *Proc. Third Int. Joint Conf. on Artificial Intelligence*, Stanford U., 1973.

30. Enea, Horace, Kenneth Mark Colby, **Idiolectic Language-Analysis for Understanding Doctor-Patient Dialogues**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

31. Faught, William S., **Affect as Motivation for Cognitive and Conative Processes**, *Adv. Papers of 4th Int. Joint Conference on Artificial Intelligence*, Vol. 2, pp. 893-899, September 1975.

32. Feldman, Jerome A., James R. Low, **Comment on Brent's Scatter Storage Algorithm**, *Comm. ACM*, November 1973.

33. Feldman, Jerome A., Yoram Yakimovsky, **Decision Theory and Artificial Intelligence: I A Semantics-based Region Analyzer**, *Artificial Intelligence J.*, Vol. 5, No. 4, Winter 1974.

34. Finkel, Raphael, Russell Taylor, Robert Bolles, Richard Paul, Jerome Feldman, **An Overview of AL, a Programming System**

for Automation, *Adv. Papers of 4th Int. Joint Conference on Artificial Intelligence*, Vol. 2, pp. 758-765, September 1975.

35. Floyd, Robert, Louis Steinberg, An Adaptive Algorithm for Spatial Greyscale, *Proc. Society for Information Display*, Volume 17, Number 2, pp. 75-77, Second Quarter 1976.

36. Fuller, Samuel H., Forest Baskett, An Analysis of Drum Storage Units, *J. ACM*, Vol. 22, No. 1, January 1975.

37. Funt, Brian, WHISPER: A Problem-solving System utilizing Diagrams and a Parallel Processing Retina, *Proc. Int. Joint Conf. on A.I.*, August 1977.

38. Gennery, Don A Stereo Vision System for an Autonomous Vehicle, *Proc. Int. Joint Conf. on A.I.*, August 1977.

39. Gennery, D.B., A Stereo Vision System for Autonomous Vehicles, *Proc. Image Understanding Workshop*, Palo Alto, Oct 1977.

40. German, Steven, Automating Proofs of the Absence of Common Runtime Errors, *Proc. 5th ACM Symposium on Principles of Programming Languages*, pp. 105-118, January 1978.

41. Goad, Chris, Monadic Infinitary Propositional Logic: A Special Operator, *Reports on Mathematical Logic*, 10, 1978.

42. Goad, Chris, Proofs as descriptions of computations, *Proc. Automatic Deduction Conference*, Les Arcs (France), July 8-11, 1980.

43. Goldman, Neil M., Sentence Paraphrasing from a Conceptual Base, *Comm. ACM*, February 1975.

44. Goldman, Ron, Recent Work with the AL System, *Proc. Int. Joint Conf. on A.I.*, August 1977.

45. Green, Cordell, David Barstow, Some Rules for the Automatic Synthesis of Programs, *Adv. Papers of 4th Int. Joint Conference on Artificial Intelligence*, Vol. 1, pp. 232-239, September 1975.

46. Green, Cordell, and Barstow, David, Some Rules for the Automatic Synthesis of Programs, *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Volume 1, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1975, pages 232-239.

47. Green, Cordell, The Design of the PSI Program Synthesis System, *Proc. 2nd Int. Conf. on Software Engineering*, IEEE Computer Society, Long Beach, California, October 1976.

48. Green, Cordell, The PSI Program Synthesis System, 1976, *ACM '76: Proceedings of the Annual Conference*, Association for Computing Machinery, New York, New York, October 1976, pages 74-75.

49. Green, C. C., and Barstow, D. R., A Hypothetical Dialogue Exhibiting a Knowledge Base for a Program Understanding System, in Elcock, E. W., and Michie, D., editors, *Machine Intelligence 8: Machine Representations of Knowledge*, Ellis Horwood, Ltd., and John Wiley and Sons, Inc., New York, New York, 1976.

50. Green, C. C., A Summary of the PSI Program Synthesis System, *Proc. Int. Joint Conf. on A.I.*, August 1977.

51. Harvey, Brian, Increasing Programmer Power at Stanford with Display Terminals, *Minutes of the DECsystem-10 Spring-75 DECUS Meeting*, Digital Equipment Computer Users Society, Maynard, Mass., 1975.

52. Hieronymus, J. L., N. J. Miller, A. L. Samuel, The Amanuensis Speech Recognition System, *Proc. IEEE Symposium on Speech Recognition*, April 1974.

53. Hieronymus, J. L., Pitch Synchronous Acoustic Segmentation, *Proc. IEEE Symposium on Speech Recognition*, April 1974.

54. Hilf, Franklin, Use of Computer Assistance in Enhancing Dialog Based Social Welfare, Public Health, and Educational Services in Developing Countries, *Proc. 2nd Jerusalem Conf. on Info. Technology*, July 1974.

55. Hilf, Franklin, Dynamic Content Analysis, *Archives of General Psychiatry*, January 1975.

56. Hueckel, Manfred H., A Local Visual Operator which Recognizes Edges and Lines, *J. ACM*, October 1973.

57. Igarashi, S., R. L. London, D. C. Luckham, Automatic Program Verification I: Logical Basis and its Implementation, *Acta Informatica*,, Vol. 4, pp.145-182, March 1975.

58. Ishida, Tatsuzo, Force Control in Coordination of Two Arms, *Proc. Int. Joint Conf. on A.I.*, August 1977.

59. Kant, Elaine, The Selection of Efficient Implementations for a High-level Language, *Proc. SIGART-SIGPLAN Symp. on A.I. & Prog. Lang.*, August 1977.

60. Karp, Richard A., David C Luckham, Verification of Fairness in an Implementation of Monitors, *Proc. 2nd Intnl. Conf. on Software Engineering*, PP. 40-46, October 1976.

61. Katz, Shmuel, Zohar Manna, A Heuristic Approach to Program Verification, *Proc. Third Int. Joint Conf. on Artificial Intelligence*, Stanford University, pp. 500-512, August 1973.

62. Katz, Shmuel, Zohar Manna, Towards Automatic Debugging of Programs, *Proc. Int. Conf. on Reliable Software*, Los Angeles, April 1975.

63. Katz, Shmuel, Zohar Manna, Logical Analysis of Programs, *Comm. ACM*, Vol. 19, No. 4, pp. 188-206, April 1976.

64. Katz, Shmuel, Zohar Manna, A Closer Look at Termination, *Acta Informatica*, Vol. 5, pp. 333-352, April 1977.

65. Lenat, Douglas B., BEINGS: Knowledge as Interacting Experts, *Adv. Papers of 4th Int. Joint Conference on Artificial Intelligence*, Vol. 1, pp. 126-133, September 1975.

66. Luckham, David C., Automatic Problem Solving, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

67. Luckham, David C., Jack R. Buchanan, Automatic Generation of Programs Containing Conditional Statements, *Proc. AISB Summer Conference*, U. Sussex, July 1974.

68. Luckham, David C., Nori Suzuki, Proof of Termination within a Weak Logic of Programs, *Acta Informatica*, Vol 8, No. 1, pp. 21-36, March 1977.

69. Luckham, David C., Program Verification and Verification-oriented Programming, *Proc. I.F.I.P. Congress '77*, August 1977.

70. Manna, Zohar, Program Schemas, *in Currents in the Theory of Computing* (A. V. Aho, Ed.), Prentice-Hall, Englewood Cliffs, N. J., 1973.

71. Manna, Zohar, Stephen Ness, Jean Vuillemin, **Inductive Methods for Proving Properties of Programs**, *Comm. ACM*, Vol. 16, No. 8, pp. 491-502, August 1973.

72. Manna, Zohar, **Automatic Programming**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

73. Manna, Zohar, *Mathematical Theory of Computation*, McGraw-Hill, New York, 1974.

74. Manna, Zohar, Amir Pneuli, **Axiomatic Approach to Total Correctness**, *Acta Informatica*, Vol. 3, pp. 243-263, 1974.

75. Manna, Zohar, Richard Waldinger, **Knowledge and Reasoning in Program Synthesis**, *Artificial Intelligence*, Vol. 6, pp. 175-208, 1975.

76. Manna, Zohar, Adi Shamir, **The Theoretical Aspects of the Optimal Fixpoint**, *SIAM Journal of Computing*, Vol. 5, No. 3, pp.414-426, September 1976.

77. Manna, Zohar, Richard Waldinger, **The Automatic Synthesis of Recursive Programs**, *Proc. SIGART-SIGPLAN Symp. on A.I. & Prog. Lang.*, August 1977.

78. Manna, Zohar, Richard Waldinger, **The Automatic Synthesis of Systems of Recursive Programs**, *Proc. Int. Joint Conf. on A.I.*, August 1977.

79. Manna, Zohar, Adi Shamir, **The Optimal-Fixpoint Approach to Recursive Programs**, *Comm. ACM*, Vol. 20, No. 11, pp. 824-831, November 1977.

80. Manna, Zohar, Richard Waldinger, (eds.), *Studies in Automatic Programming Logic*, American Elsevier, New York, NY, 1977.

81. Manna, Zohar, Richard Waldinger, **Is 'Sometime' sometimes better than 'Always'? Intermittant Assertions in Proving Program Correctness**, *Comm. ACM*, Vol. 21, No. 2, pp. 159-172, February 1978.

82. Manna, Zohar, Adi Shamir, **The Convergence of Functions to Fixpoints of Recursive Definitions**, *Theoretical Conputer Science J.*, Vol. 6, pp. 109-141, March 1978.

83. Manna, Zohar, Richard Waldinger, **The Logic of Computer Programming**, *IEEE Trans. Software Eng.*, Vol. SE-4, No. 5, pp. 199-224, May 1978.

84. Manna, Zohar, Richard Waldinger, **The Synthesis of Structure-changing Programs**, *Proc. 3rd Int. Conf. on Software Eng.*, Atlanta, GA, May 1978.

85. Manna, Zohar, Richard Waldinger, **The DEDALUS System**, *Proc. National Computer Conf.*, Anaheim, CA, June 1978.

86. McCarthy, John, **Mechanical Servants for Mankind**, *Britannica Yearbook of Science and the Future*, 1973.

87. McCarthy, John, Book Review: **Artificial Intelligence: A General Survey** by Sir James Lighthill, *Artificial Intelligence*, Vol. 5, No. 3, Fall 1974.

88. McCarthy, John, **Modeling Our Minds** *Science Year 1975*, The World Book Science Annual, Field Enterprises Educational Corporation, Chicago, 1974.

89. McCarthy, John, **Proposed Criterion for a Cipher to be Probable-word-proof**, *Comm. ACM*, February 1975.

90. McCarthy, John, **An Unreasonable Book**, a review of *Computer Power and Human Reason* by Joseph Weizenbaum (W.H. Freeman and Co., San Francisco, 1976), *SIGART Newsletter* •58, June 1976.

91. McCarthy, John, Review: *Computer Power and Human Reason*, by Joseph Weizenbaum (W.H. Freeman and Co., San Francisco, 1976) in *Physics Today*, 1977.

92. McCarthy, John, Another SAMEFRINGE, *SIGART Newsletter* No. 61, February 1977.

93. McCarthy, John, The Home Information Terminal, *The Grolier Encyclopedia*, 1977.

94. McCarthy, John, M. Sato, T. Hayashi, S. Igarashi, On the Model Theory of Knowledge, *Proc. Int. Joint Conf. on A.I.*, August 1977.

95. McCarthy, John, Epistemological Problems of Artificial Intelligence, *Proc. Int. Joint Conf. on A.I.*, August 1977.

96. McCarthy, John, History of LISP, *Proc. ACM Conf. on History of Programming Languages*, 1978.

97. McCarthy, John, Representation of Recursive Programs in First Order Logic, *Proc. Int. Conf. on Mathematical Studies of Information Processing*, Kyoto Japan, 1978.

98. McCarthy, John, Ascribing Mental Qualities to Machines, *Philosophical Perspectives in Artificial Intelligence*, Martin Ringle (ed.), Humanities Press, 1978.

99. McCune, Brian, The PSI Program Model Builder: Synthesis of Very High-level Programs, *Proc. SIGART-SIGPLAN Symp. on A.I. & Prog. Lang.*, August 1977.

100. Miller, N. J., Pitch Detection by Data Reduction, *Proc. IEEE Symposium on Speech Recognition*, April 1974.

101. Moore, Robert C., Reasoning about Knowledge and Action, *Proc. Int. Joint Conf. on A.I.*, August 1977.

102. Moorer, James A., The Optimum Comb Method of Pitch Period Analysis of Continuous Speech, *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. ASSP-22, No. 5, October 1974.

103. Moorer, James A., On the Transcription of Musical Sound by Computer, *USA-JAPAN Computer Conference*, August 1975.

104. Morales, Jorge J., Interactive Theorem Proving, *Proc. ACM National Conference*, August 1973.

105. Moravec, Hans, Towards Automatic Visual Obstacle Avoidance, *Proc. Int. Joint Conf. on A.I.*, August 1977.

106. Moravec, Hans, A Non-Synchronous Orbital Skyhook, *J. Astronautical Sciences*, Vol. XXV •4, pp. 307-322, Oct.-Dec. 1977.

107. Moravec, Hans, Today's Computers, Intelligent Machines and Our Future, *Analog*, Vol. 99 •2, pp. 59-84, February 1979.

108. Moravec, Hans P., Visual Mapping by a Robot Rover, *Proc. 6th Int. Joint Conf. on Artificial Intelligence*, Tokyo, Japan, August 1979, pp. 589-601.

109. Moravec, Hans, A Non-synchronous Orbital Skyhook, *J. Astronautical Sciences*, Vol 26, No. 1, 1978.

110. Moravec, Hans P., Fully Interconnecting Multiple Computers with Pipelined Sorting Nets, *IEEE Trans. on Computers*, October 1979.

111. Moravec, Hans, Cable Cars in the Sky, *The Endless Frontier*, Vol. 1, Jerry Pournelle, ed., Grosset & Dunlap, Ace books, November 1979.

112. Moravec, Hans, The Endless Frontier and The Thinking Machine, *The Endless Frontier*, Vol. 2, Jerry Pournelle, ed., Grosset & Dunlap, Ace books, 1980. (to appear)

113. Nelson, C. G., Oppen, D. C., Fast Decision Algorithms based on UNION and FIND, *Proc. 18th Annual IEEE Symposium on Foundations of Computer Science*, October 1977.

114. Nelson, C. G., Derek Oppen, **A Simplifier based on Fast Decision Algorithms**, *Proc. Fifth ACM Symposium on Principles of Programming Languages*, January 1978.

115. Nelson, C.G. and Oppen, D., **Simplification by Cooperating Decision Procedures**, *ACM Transactions on Programming Languages and Systems*, Oct. 1979.

116. Nevatia, Ramakant, Thomas O. Binford, **Structured Descriptions of Complex Objects**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

117. Nevatia, R., T.O. Binford; **Structured Descriptions of Complex Objects**; *Artificial Intelligence*, 1977.

118. Newell, A., Cooper, F. S., Forgie, J. W., Green, C. C., Klatt, D. H., Medress, M. F., Neuburg, E. P., O'Malley, M. H., Reddy, D. R., Ritea, B., Shoup, J. E., Walker, D. E., and Woods, W. A., *Considerations for a Follow-On ARPA Research Program for Speech Understanding Systems*, Information Processing Techniques Office, Advanced Research Projects Agency, Department of Defense, Arlington, Virginia, August 1975.

119. Oppen, Derek, S.A. Cook, **Proving Assertions about Programs that Manipulate Data Structures**, *Acta Informatica*, Vol. 4, No. 2, pp. 127-144, 1975.

120. Oppen, Derek C., **Reasoning about Recursive Data Structures**, *Proc. Fifth ACM Symposium on Principles of Programming Languages*, January 1978.

121. Oppen, Derek C., **A Superexponential Bound on the Complexity of Presburger Arithmetic**, *Journal of Computer and Systems Sciences*, June 1978.

122. Oppen, Derek, **Convexity, Complexity, and Combinations of Theories**, *Proc. 5th Symposium on Automated Deduction*, January 1979.

123. Phillips, Jorge, T. H. Bredt, **Design and Verification of Real-time Systems**, *Proc. 2nd Int. Conf. on Software Engineering*, IEEE Computer Society, Long Beach, California, October 1976.

124. Phillips; Jorge, **Program Inference from Traces using Multiple Knowledge Sources**, *Proc. Int. Joint Conf. on A.I.*, August 1977.

125. Phillips, Jorge V., **LISP and Its Semantics**, *Comunicaciones Tecnicas* (in Spanish), Blue Series: monographs, Center for Research in Applied Mathematics and Systems, National University of Mexico, Mexico City, Mexico, June 1978.

Phillips, Jorge V., **Abstract Programming**, *Colombia Electronica* (in Spanish), Volume IV-2 1979, Apartado Aereo 1825, Bogota, Colombia, June 1979.

126. Polak, Wolfgang, **An exercise in automatic program verification**, *IEEE Trans. on Software Engineering*, vol. SE-5, Sept. 1979.

127. Quam, Lynn, Robert Tucker, Botond Eross, J. Veverka and Carl Sagan, **Mariner 9 Picture Differencing at Stanford**, *Sky and Telescope*, August 1973.

128. Rubin, Jeff, **Computer Communication via the Dial-up Network**, *Minutes of the DECsystem-10 Spring-75 DECUS Meeting*, Digital Equipment Computer Users Society, Maynard, Mass., 1975.

129. Sagan, Carl, J. Veverka, P. Fox, R. Dubisch, R. French, P. Gierasch, L. Quam, J. Lederberg, E. Levinthal, R. Tucker, B. Eross, J. Pollack, **Variable Features on Mars II: Mariner 9 Global Results**, *J. Geophys. Res.*, 78, 4163-4196, 1973.

130. Samet, Hanan, **Proving the Correctness of Heuristically Optimized Code**, *Comm. ACM*, July 1978.

131. Schank, Roger C., Neil Goldman, Charles J. Rieger III, Chris Riesbeck, **MARGIE: Memory, Analysis, Response Generation and Inference on English**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

132. Schank, Roger C., Kenneth Colby (eds), *Computer Models of Thought and Language*, W. H. Freeman, San Francisco, 1973.

133. Schank, Roger, **The Conceptual Analysis of Natural Language**, in R. Rustin (ed.), *Natural Language Processing*, Algorithmics Press, New York, 1973.

134. Schank, Roger, Charles J. Rieger III, **Inference and Computer Understanding of Natural Language**, *Artificial Intelligence J.*, Vol.5, No. 4, Winter 1974.

135. Schank, Roger C., Neil M. Goldman, Charles J. Rieger III, Christopher K. Riesbeck, **Interface and Paraphrase by Computer**, *J. ACM*, Vol 22, No. 3, July 1975.

136. Shaw, David E., William R. Swartout, C. Cordell Green, **Inferring LISP Programs from Examples**, *Adv. Papers of 4th Int. Joint Conference on Artificial Intelligence*, Vol. 1, pp. 260-267, September 1975.

137. Shortliffe, Edward H., Davis, Randall, Axline, Stanton G., Buchanan, Bruce G., Green, C. Cordell, and Cohen, Stanley N., **Computer-Based Consultations in Clinical Therapeutics: Explanation and Rule Acquisition Capabilities of the MYCIN System**, *Computers and Biomedical Research*, Volume 8, Number 3, June 1975, pages 303-320.

138. Smith, David Canfield, Horace J. Enea, **Backtracking in MLISP2**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

139. Smith, Leland, **Editing and Printing Music by Computer**, *J. Music Theory*, Fall 1973.

140. Sobel, Irwin, **On Calibrating Computer Controlled Cameras for Perceiving 3-D Scenes**, *Proc. Third Int. Joint Conf. on Artificial Intelligence*, Stanford U., 1973; also in Artificial Intelligence J., Vol. 5, No. 2, Summer 1974.

141. Suzuki, N., **Verifying Programs by Algebraic and Logical Reduction**, *Proc. Int. Conf. on Reliable Software*, Los Angeles, Calif., April 1975, in *ACM SIGPLAN Notices*, Vol. 10, No. 6, pp. 473-481, June 1975.

142. Tesler, Lawrence G., Horace J. Enea, David C. Smith, **The LISP70 Pattern Matching System**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

143. Thomas, Arthur J., **Puccetti on Machine Pattern Recognition**, *Brit.J.Philosophy of Science*, 26:227-232, 1975.

144. Tracy, Kathleen B., Elaine C. Montague, Richard P. Gabriel, Barbara E. Kent, **Computer-Assisted Diagnosis of Orthopedic Gait Disorders**, *Physical Therapy*, Vol. 59, No. 3., pp 268-277, March 1979.

145. Veverka, J., Carl Sagan, Lynn Quam, R. Tucker, B. Eross, **Variable Features on Mars III: Comparison of Mariner 1969 and Mariner 1971 Photography**, *Icarus*, 21, 317-368, 1974.

146. von Henke, F. W., D.C. Luckham, **A Methodology for Verifying Programs**, *Proc. Int. Conf. on Reliable Software*, Los Angeles, Calif., April 1975, in *ACM SIGPLAN Notices*, Vol. 10, No. 6, pp. 156-164, June 1975.

147. Wilks, Yorick, **The Stanford Machine Translation and Understanding Project**, in R. Rustin (ed.), *Natural Language Processing*, Algorithmics Press, New York, 1973.

148. Wilks, Yorick, **Understanding Without Proofs**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

149. Wilks, Yorick, Annette Herskovits, **An Intelligent Analyser and Generator of Natural Language**, *Proc. Int. Conf. on Computational Linguistics*, Pisa, Italy, *Proceedings of the Third Internation Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

150. Wilks, Yorick, **The Computer Analysis of Philosophical Arguments**, *CIRPHO*, Vol. 1, No. 1, September 1973

151. Wilks, Yorick, **An Artificial Intelligence Approach to Machine Translation**, in Schank and Colby (eds.), *Computer Models of Thought and Language*, W. H. Freeman, San Francisco, 1973.

152. Wilks, Yorick, **One Small Head – Models and Theories in Linguistics**, *Foundations of Language*, Vol. 10, No. 1, January 1974.

153. Wilks, Yorick, **Preference Semantics**, E. Keenan (ed.), *Proc. 1973 Colloquium on Formal Semantics of Natural Language*, Cambridge, U.K., 1974.

154. Wilks, Yorick, **The XGP Computer-driven Printer at Stanford**, *Bulletin of Assoc. for Literary and Linguistic Computing*, Vol. 2, No. 2, Summer 1974.

155. Wilks, Y., **Semantic Procedures and Information**, in *Studies in the Foundations of Communication*, R. Posner (ed.), Springer, Berlin, forthcoming.

156. Wilks, Yorick, **A Preferential, Pattern-Seeking Semantics for Natural Language Inference**, *Artificial Intelligence J.*, Vol. 6, No. 1, Spring 1975.

157. Wilks, Y., **An Intelligent Analyser and Understander of English**, *Comm. ACM*, May 1975.

158. Winograd, Terry, **A Process Model of Language Understanding**, in Schank and Colby (eds.), *Computer Models of Thought and Language*, W. H. Freeman, San Francisco, 1973.

159. Winograd, Terry, **The Processes of Language Understanding** in Benthall, (ed.), *The Limits of Human Nature*, Allen Lane, London, 1973.

160. Winograd, Terry, **Language and the Nature of Intelligence**, in G.J. Dalenoort (ed.), *Process Models for Psychology*, Rotterdam Univ. Press, 1973

161. Winograd, Terry, **Breaking the Complexity Barrier (again)**, *Proc. SIGPLAN-SIGIR Interface Meeting*, 1973; *ACM SIGPLAN Notices*, 10:1, pp. 13-30, January 1975.

162. Winograd, Terry, **Artificial Intelligence – When Will Computers Understand People?**, *Psychology Today*, May 1974.

163. Winograd, Terry, **Frame Representations and the Procedural - Declarative Controversy**, in D. Bobrow and A. Collins, eds., *Representation and Understanding: Studies in Cognitive Science*, Academic Press, 1975.

164. Winograd, Terry, **Reactive Systems**, *Coevolution Quarterly*, September 1975

165. Winograd, Terry, **Parsing Natural Language via Recursive Transition Net**, in Raymond Yeh (ed.) *Applied Computation Theory*, Prentice-Hall, 1976.

166. Winograd, Terry, **Computer Memories – a Metaphor for Human Memory**, in Charles Cofer (ed.), *Models of Human Memory*, Freeman, 1976.

167. Yakimovsky, Yoram, Jerome A. Feldman, **A Semantics-Based Decision Theoretic Region Analyzer**, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford University, August 1973.

168. Yolks, Warwick, **There's Always Room at the Top, or How Frames gave my Life Meaning**, *SIGART Newsletter*, No. 53, August 1975.

Appendix D
Abstracts of Recent Reports

Abstracts are given here for Artificial Intelligence Memos published since 1976. For earlier years, see our ten-year report [Memo AIM-228] or diskfile AIMS.OLD [BIB,DOC] @SU-AI. The abstracts below are kept in diskfile AIMS [BIB,DOC] @SU-AI and the titles of both earlier and more recent A. I. Memos are in AIMLST[BIB,DOC] @SU-AI.

In the listing below, there are up to three numbers given for each report: an "AIM" number on the left, a "CS" (Computer Science) number in the middle, and a NTIS stock number (often beginning "AD...") on the right. Special symbols preceding the "AIM" number indicate availability at this writing, as follows:

+ hard copy or microfiche,
- microfiche only,
* out-of-stock.

If there is no special symbol, then it is available in hard copy only. Reports that are in stock may be requested from:

Documentation Services
Artificial Intelligence Laboratory
Stanford University
Stanford, California 94305

Rising costs and restrictions on the use of research funds for printing reports have made it necessary to charge for reports at their replacement cost. By doing so, we will be able to reprint popular reports rather than simply declaring them "out of print".

Alternate Sources

Alternatively, reports may be ordered (for a nominal fee) in either hard copy or microfiche from:

National Technical Information Service
P. O. Box 1553
Springfield, Virginia 22161

If there is no NTIS number given, then they may or may not have the report. In requesting copies in this case, give them both the "AIM-" and "CS-nnn" numbers, with the latter enlarged into the form "STAN-CS-yy-nnn", where "yy" is the last two digits of the year of publication.

Memos that are also Ph.D. theses are so marked below and may be ordered from:

University Microfilm
P. O. Box 1346
Ann Arbor, Michigan 48106

For people with access to the ARPA Network, the texts of some A. I. Memos are stored online in the Stanford A. I. Laboratory disk file. These are designated below by "Diskfile: <file name>" appearing in the header.

- AIM-277 CS-542 ADA027454
Zohar Manna, Adi Shamir,
The Theoretical Aspects of the Optimal Fixedpoint,
24 pages, March 1976.

In this paper we define a new type of fixedpoint of recursive definitions and investigate some of its properties. This optimal fixedpoint (which always uniquely exists) contains, in some sense, the maximal amount of "interesting" information which can be extracted from the recursive definition, and it may be strictly more defined than the program's least fixedpoint. This fixedpoint can be the basis for assigning a new semantics to recursive programs.

+ AIM-278 CS-549 ADA027455
David Luckham, Norihisa Suzuki,
Automatic Program Verification V: Verification-Oriented Proof Rules for Arrays, Records and Pointers,
48 pages, March 1976. Cost: $3.05

A practical method is presented for automating in a uniform way the verification of Pascal programs that operate on the standard Pascal data structures ARRAY, RECORD, and POINTER. New assertion language primitives are introduced for describing computational effects of operations on these data structures. Axioms defining the semantics of the new primitives are given. Proof rules for standard

Pascal operations on pointer variables are then defined in terms of the extended assertion language. Similar rules for records and arrays are special cases. An extensible axiomatic rule for the Pascal memory allocation operation, NEW, is also given.

These rules have been implemented in the Stanford Pascal program verifier. Examples illustrating the verification of programs which operate on list structures implemented with pointers and records are discussed. These include programs with side-effects.

- AIM-279 CS-552
Norihisa Suzuki,
Automatic Verification of Programs with Complex Data Structures,
Thesis: Ph.D. in Computer Science,
194 pages, February 1976.

The problem of checking whether programs work correctly or not has been troubling programmers since the earliest days of computing. Studies have been conducted to formally define semantics of programming languages and derive proof rules for correctness of programs.

Some experimental systems have been built to mechanically verify programs based on these proof rules. However, these systems are yet far from attacking real programs in a real environment. Many problems covering the ranges from theory to artificial intelligence and programming languages must be solved in order to make program verification a practical tool. First, we must be able to verify a complete practical programming language. One of the important features of real programming languages which is not treated in early experimental systems is complex data structures. Next, we have to study specification methods. In order to verify programs we have to express what we intend to do by the programs. In many cases we are not sure what we want to verify and how we should express them. These specification methods are not independent of the proof rules. Third, we have to construct an efficient prover so that we can interact with the verification process. It is expected that repeated verification attempts will be necessary because programs and specifications may have errors at first try. So the time to complete one verification attempt is very important in real environment.

We have chosen Pascal as the target language. The semantics and proof rules are studied by Hoare & Wirth and Igarashi, London & Luckham. However, they have not treated complex data structures obtained from arrays, records, and pointers. In order to express the state of the data structures concisely and express the effects of statements we introduced special assertion language primitives and new proof rules. We defined new methods of introducing functions and predicates to write assertions so that we can express simplification rules and proof search strategies. We introduced a special language to document properties of these functions and predicates. These methods enable users to express assertions in natural ways so that verification becomes easier. The theorem prover is constructed so that it will be efficient for proving a type of formulas which appear very often as verification conditions.

We have successfully verified many programs. Using our new proof rules and specification methods we have proved properties of sorting programs such as permutation and stability which have been thought to be hard to prove. We see no theoretical as well as practical problems in verifying sorting programs. We have also verified programs which manipulate pointers. These programs change their data structures so that usually verification conditions tend to be complex and hard to read. Some study about the complexity problem seems necessary.

The verifier has been used extensively by various users, and probably the most widely used verifier implemented so far. There is yet a great deal of research necessary in order to fill the gap between the current verifier and the standard programming tools like editors and compilers.

This dissertation was submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

+ AIM-280 CS-555
David D. Grossman,
Monte Carlo Simulation of Tolerancing in Discrete Parts Manufacturing and Assembly,
25 pages, May 1976. Cost: $2.40

The assembly of discrete parts is strongly affected by imprecise components, imperfect fixtures and tools, and inexact measuremets. It is often necessary to design higher precision into the manufacturing and assembly process than is functionally needed in the final product. Production engineers must trade off between alternative ways of selecting individual tolerances in order to achieve minimum cost, while preserving product integrity. This paper describes a comprehensive Monte Carlo method for systematically analysing the stochastic implications of tolerancing and related forms of imprecision. The method is illustrated by four examples, one of which is chosen from the field of assembly by computer controlled manipulators.

+ AIM-281.1 CS-558 AD-A042 507
Zohar Manna, Richard Waldinger,
Is 'sometime' sometimes better than 'always'? Intermittent assertions in proving program correctness,
41 pages, June 1976, revised March 1977. Cost: $2.85

This paper explores a technique for proving the correctness and termination of programs simultaneously. This approach, which we call the [intermittent]-[assertion method], involves documenting the program with assertions that must be true at some time when control is passing through the corresponding point, but that need not be true every time. The method, introduced by Knuth and further developed by Burstall, promises to provide a valuable complement to the more conventional methods. We first introduce and illustrate the technique with a number of examples. We then show that a correctness proof using the invariant assertion method or the subgoal induction method can always be expressed using intermittent assertions instead, but that the reverse is not always the case. The method can also be used just to prove termination, and any proof of termination using the conventional well-founded sets approach can be rephrased as a proof using intermittent assertions. Finally, we show how the method can be applied to prove the validity of program transformations and the correctness of continuously operating programs.

This is a revised and simplified version of a pevious paper with the same title (AIM-281, June 1976).

+ AIM-282 CS-560
Russell Taylor,
Synthesis of Manipulator Control Programs from Task-level Specifications,
Thesis: Ph.D. in Computer Science,
229 pages, July 1976. Cost: $8.10

This research is directed towards automatic generation of manipulator control programs from task-level specifications. The central assumption is that much manipulator-level coding is a process of adapting known program constructs to particular tasks, in which coding decisions are made by well-defined computations based on *planning information.* For manipulator programming, the principal elements of planning information are: (1) descriptive information about the objects being manipulated; (2) situational information describing the execution-time environment; and (3) action information defining the task and the semantics of the execution-time environment.

A standard subtask in mechanical assembly, insertion of a pin into a hole, is used to focus the technical issues of automating manipulator coding decisions. This task is first analyzed from the point of view of a human programmer writing in the target language, AL, to identify the specific coding decisions required

and the planning information required to make them. Then, techniques for representing this information in a computationally useful form are developed. Objects are described by attribute graphs, in which the nodes contain shape information, the links contain structural information, and properties of the links contain location information. Techniques are developed for representing object locations by parameterized mathematical expressions in which free scalar variables correspond to degrees of freedom and for deriving such descriptions from symbolic relations between object features. Constraints linking the remaining degrees of freedom are derived and used to predict maximum variations. Differential approximations are used to predict errors in location values. Finally, procedures are developed which use this planning information to generate AL code automatically.

The AL system itself performs a number of coding functions not normally found in algebraic compilers. These functions and the planning information required to support them are also discussed.

- AIM-283 CS-552
Randall Davis,
Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases,
Thesis: Ph.D. in Computer Science,
304 pages, July 1976.

The creation and management of large knowledge bases has become a central problem of artificial intelligence research as a result of two recent trends: an emphasis on the use of large stores of domain specific knowledge as a base for high performance programs, and a concentration on problems taken from real world settings. Both of these mean an emphasis on the accumulation and management of large collections of knowledge, and in many systems embodying these trends much time has been spent on building and maintaining such knowledge bases. Yet there has been little discussion or analysis of the concomitant problems. This thesis attempts to define some of the issues involved, and explores steps taken toward solving a number of the problems encountered. It describes the organization, implementation, and operation of a program called TEIRESIAS, designed to make possible the interactive transfer of expertise from a human expert to the knowledge base of a high performance program, in a dialog conducted in a restricted subset of natural language.

The two major goals set were (*i*) to make it possible for an expert in the domain of application to "educate" the performance program directly, and (*ii*) to ease the task of assembling and maintaining large amounts of knowledge.

The central theme of this work is the exploration and use of what we have labelled *meta level knowledge*. This takes several different forms as its use is explored, but can be summed up generally as "knowing what you know". It makes possible a system which has both the capacity to use its knowledge directly, and the ability to examine it, abstract it, and direct its application.

We report here on the full extent of the capabilities it makes possible, and document cases where its lack has resulted in significant difficulties. Chapter 3 describes efforts to enable a program to explain its actions, by giving it a model of its control structure and an understanding of its representations. Chapter 5 documents the use of abstracted models of knowledge (*rule models*) as a guide to acquisition. Chapter 6 demonstrates the utility of describing to a program the structure of its *representations (using data structure schemata)*. Chapter 7 describes the use of strategies in the form of *meta rules*, which contain knowledge about the use of knowledge.

- AIM-284 CS-567
Rafael Finkel,
Constructing and Debugging Manipulator Programs,
Thesis: Ph.D. in Computer Science,
171 pages pages, August 1976.

This thesis presents results of work done at the Stanford Artificial Intelligence Laboratory in the field of robotics. The goal of the work is to program mechanical manipulators to accomplish a range of tasks, especially those found in the context of automated assembly. The thesis has three chapters describing significant work in this domain. The first chapter is a textbook that lays a theoretical framework for the principal issues involved in computer control of manipulators, including types of manipulators, specification of destinations, trajectory specification and planning, methods of interpolation, force feedback, force application, adaptive control, collision avoidance, and simultaneous control of several manipulators. The second chapter is an implementation manual for the AL manipulator programming language. The goals of the language are discussed, the language is defined, the compiler described, and the execution environment detailed. The language has special facilities for condition monitoring, data types that represent coordinate systems, and affixment structures that allow coordinate systems to be linked together. Programmable side effects play a large role in the implementation of these features. This chapter closes with a detailed programming example that displays how the constructs of the language assist in formulating and encoding the manipulation task. The third chapter discusses the problems involved in programming in the AL language, including program preparation, compilation, and especially debugging. A debugger, ALAID, is designed to make use of the complex environment of AL. Provision is made to take advantage of the multiple-processor, multiple-process, real-time, interactive nature of the problem. The principal conclusion is that the debugger can fruitfully act as a uniform supervisor for the entire process of program preparation and as the means of communication between cooperating processors.

- AIM-285 CS-568 PB-259 130/3WC
T. O. Binford, D. D. Grossman, C. R. Lui, R. C. Bolles, R. A. Finkel, M. S. Mujtaba, M. D. Roderick, B. E. Shimano, R. H. Taylor, R. H. Goldman, J. P. Jarvis, V. D. Scheinman, T. A. Gafford,
Exploratory Study of Computer Integrated Assembly Systems, Progress Report 3,
336 pages, August 1976.

The Computer Integrated Assembly Systems project is concerned with developing the software technology of programmable assembly devices, including computer controlled manipulators and vision systems. A complete hardware system has been implemented that includes manipulators with tactile sensors and TV cameras, tools, fixtures, and auxiliary devices, a dedicated minicomputer, and a time-shared large computer equipped with graphic display terminals. An advanced software system call AL has been developed that can be used to program assembly applications. Research currently underway includes refinement of AL, development of improved languages and interactive programming techniques for assembly and vision, extension of computer vision to areas which are currently infeasible, geometric modeling of objects and constraints, assembly simulation, control algorithms, and adaptive methods of calibration.

+ AIM-285.4 CS-568 PB-259 130/3WC
T. O. Binford, C. R. Lui, G. Gini, M. Gini, I. Glaser, T. Ishida, M. S. Mujtaba, E. Nakano, H. Nabavi, E. Panofsky, B. E. Shimano, R. Goldman, V. D. Scheinman, D. Schmelling, T. A. Gafford,
Exploratory Study of Computer Integrated Assembly Systems, Progress Report 4,
255 pages, June 1977. Cost: $8.85

The Computer Integrated Assembly Systems project is concerned with developing the software technology of programmable assembly devices. A primary part of the research has been designing and building the AL language for assembly. A first level version of AL is now implemented and debugged, with user

interfaces. Some of the steps involved in completing the system are described. The AL parser has been completed and is documented in this report. A preliminary interface with vision is in operation. Several hardware projects to support software development have been completed. One of the two Stanford arms has been rebuilt. An electronic interface for the other arm has been completed. Progress on other hardware aspects of the AL systems is reported.

Several extensions to AL are described. A new interactive program for building models by teaching is running and undergoing further development. Algorithms for force compliance have been derived; a software system for force compliance has been implemented and is running in the AL runtime system. New algorithms have been derived for cooperative manipulation using two manipulators. Preliminary results are described for parth calculation; these results are steps along the way to a runtime path calculator which will be important in making an export version of AL.

Results are described in analysis of several complex assemblies. These results show that two manipulators are necessary in a significant fraction of assembly operations.

Studies are described which focus on making AL meet the realities of industrial research and production use. Results of a questionaire of leading industrial and research laboratores are presented. A summary is presented of the Workshop on Softwre for Assembly, held immediately before the NSF-RANN Conference at IITRI, Nov. 1976.

- AIM-286 CS-570
Douglas Lenat,
AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search,
Thesis: Ph.D. in Computer Science,
350 pages, July 1976.

A program, called "AM", is described which models one aspect of elementary mathematics research: developing new concepts under the guidance of a large body of heuristic rules. "Mathematics" is considered as a type of intelligent behavior, not as a finished product.

+ AIM-287 CS-571
Michael Roderick,
Discrete Control of a Robot Arm,
Thesis: Engineer in Electrical Engineering,
98 pages, August 1976. Cost: $4.45

The primary goal of this thesis was to determine the feasibility of operating the Stanford robot arm and reduce sample rates. A secondary goal was to reduce the effects of variations in inertia and sampling rates on the control system's stability.

A discrete arm model was initially developed to illustrate the effects of inertia and sampling rate variations on the present control system. Modifications were then suggested for reducing these effects. Finally, a method was demonstrated for reducing the arm sampling rate from its present value of 60 hertz to approximately 45 hertz without significantly effecting the arms performance.

+ AIM-288 CS-572
Robert Filman, Richard Weyhrauch,
An FOL Primer,
36 pages, September 1976. Cost: $2.70

This primer is an introduction to FOL, an interactive proof checker for first order logic. Its examples can be used to learn the FOL system, or read independently for a flavor of our style of interactive proof checking. Several example proofs are presented, successively increasing in the complexity of the FOL commands employed.

FOL runs on the computer at the Stanford Artificial Intelligence Laboratory. It can be used over the ARPA net after arrangements have been made with Richard Weyhrauch (network address RWW@SU-AI).

+ AIM-289 CS-574
John Reiser (ed.),
SAIL,
178 pages, August 1976. Cost: $6.70

SAIL is a high-level programming language for the PDP-10 computer. It includes an extended ALGOL 60 compiler and a companion set of execution-time routines. In addition to ALGOL, the language features: (1) flexible linking to hand-coded machine language algorithms, (2) complete access to the PDP-10 I/O facilities, (3) a complete system of compile-time arithmetic and logic as well as a flexible macro system, (4) a high-level debugger, (5) records and references, (6) sets and lists, (7) an associative data structure, (8) independent processes, (9) procedure variables, (10) user modifiable error handling, (11) backtracking, and (12) interrupt facilities.

This manual describes the SAIL language and the execution-time routines for the typical SAIL user: a non-novice programmer with some knowledge of ALGOL. It lies somewhere between being a tutorial and a reference manual.

+ AIM-290 CS-575 AD-A042 494
Nancy W. Smith,
SAIL Tutorial,
54 pages, November 1976. Cost: $3.20

This TUTORIAL is designed for a beginning user of Sail, an ALGOL-like language for the PDP10. The first part covers the basic statements and expressions of the language; remaining topics include macros, records, conditional compilation, and input/output. Detailed examples of Sail programming are included throughout, and only a minimum of programming background is assumed.

- AIM-291 CS-577 AO44713
Bruce Buchanan, Joshua Lederberg, John McCarthy,
Three Reviews of J. Weizenbaum's Computer Power and Human Reason,
28 pages, November 1976.

Three reviews of Joseph Weizenbaum's **Computer Power and Human Reason** (W.H. Freeman and Co., San Francisco, 1976) are reprinted from other sources. A reply by Weizenbaum to McCarthy's review is also reprinted.

+ AIM-292 CS-580
Terry Winograd,
Towards a Procedural Understanding of Semantics,
30 pages, October 1976. Cost: $2.55

The term "procedural semantics" has been used in a variety of ways, not all compatible, and not all comprehensible. In this paper, I have chosen to apply the term to a broad paradigm for studying semantics (and in fact, all of linguistics). This paradigm has developed in a context of writing computer programs which use natural language, but it is not a theory of computer programs or programming techniques. It is "procedural" because it looks at the underlying structure of language as fundamentally shaped by the nature of processes for language production and comprehension. It is based on the belief that there is a level of explanation at which there are significant similarities between the psychological processes of human language use and the computational processes in computer programs we can construct and study. Its goal is to develop a body of theory at this level. This approach necessitates abandoning or modifying several currently accepted doctrines, including the way in which distinctions have been drawn between "semantics" and "pragmatics" and between "performance" and "competence".

The paper has three major sections. It first lays out the paradigm assumptions which guide the enterprise, and elaborates a model of cognitive processing and language use. It then illustrates how some specific semantic problems might be approached from a procedural perspective, and contrasts the procedural approach with formal structural and truth conditional approaches. Finally, it discusses the goals of linguistic theory and the nature of the linguistic explanation.

Much of waht is presented here is a speculation about the nature of a pradigm yet to be developed. This paper is an attempt to be evocative rather than definitive; to convey intuitions rather than to formulate crucial arguments which justify this approach over others. It will be successful if it suggests some ways of looking at language which lead to further understanding.

- AIM-293 CS-581 AD-A042 508
Daniel Bobrow, Terry Winograd,
An Overview of KRL,
40 pages, November 1976.

This paper describes KRL, a Knowledge Representation Language designed for use in understander systems. It outlines both the general concepts which underlie our research and the details of KRL-0, an experimental implementation of some of these concepts. KRL is an attempt to integrate procedural knowledge with a broad base of declarative forms. These forms provide a variety of ways to express the logical structure of the knowledge, in order to give flexibility in associating procedures (for memory and reasoning) with specific pieces of knowledge, and to control the relative accessibility of different facts and descriptions. The formalism for declarative knowledge is based on *structured conceptual objects* with associated *descriptions*. These objects form a network of *memory units* with several different sorts of linkages, each having well-specified implications for the retrieval process. Procedures can be associated directly with the internal structure of a conceptual object. This *procedural attachment* allows the steps for a particular operation to be determined by characteristics of the specific entities involved.

The control structure of KRL is based on the belief that the next generation of intelligent programs will integrate data-directed and goal-directed processing by using multi-processing. It provides for a priority-ordered multi-process agenda with explicit (user-provided) strategies for scheduling and resource allocation. It provides *procedure directories* which operate along with *process frameworks* to allow procedural parameterization of the fundamental system processes for building, comparing, and retrieving memory structures. Future development of KRL will include integrating procedure definition with the descriptive formalism.

+ AIM-294 CS-586 AD-A042 516
Nachum Dershowitz, Zohar Manna,
The Evolution of Programs: A System for Automatic Program Modification,
45 pages, December 1976. Cost: $2.95

An attempt is made to formulate techniques of program modification, whereby a program that achieves one result can be transformed into a new program that uses the same principles to achieve a different goal. For example, a program that uses the binary search paradigm to calculate the square-root of a number may be modified to divide two numbers in a similar manner, or vice versa.

Program debugging is considered as a special case of modification: if a program computes wrong results, it must be modified to achieve the intended results. the application of abstract *program schemata to concrete* problems is also viewed from the perspective of modification techniques.

We have embedded this approach in a running implementation; our methods are illustrated with several examples that have been performed by it.

+ AIM-295 CS-591
Robert C. Bolles,
Verification Vision Within a Programmable Assembly System,
Thesis: Ph.D. in Computer Science,
245 pages, December 1976. Cost: $8.55

The long-range goal of this research is to simplify visual information processing by computer. The research reported in this thesis concentrates on a subclass of visual information processing referred to as *verification vision* (abbreviated VV). VV includes a significant

portion of the visual feedback tasks required within programmable assembly. There are several types of information available in VV tasks that can facilitate the solution of such tasks. The main question addressed in this thesis is how to use all of this information to perform the task efficiently. Two steps are involved in answering this question: (1) formalize the types of tasks, available information, and quantities of interest and (2) formulate combination rules that use the available information to estimate the quantities of interest.

The combination rules that estimate confidences are based upon Bayes' theorem. They are general enough to handle operators that are not completely reliable, i.e., operators that may find any one of several features or a *surprise*. The combination rules that estimate precisions are based upon a least-squares technique. They use the expected precisions of the operators to check the structural consistency of a set of matches and to estimate the resulting precisions about the points of interest. An interactive VV system based upon these ideas has been implemented. It makes it possible for a person who is not an expert in vision research to program visual feedback tasks. This system helps the programmer select potentially useful operator/feature pairs, provides a training session to gather statistics on the behavior of the operators, automatically ranks the operator/feature pairs according to their expected contributions, and performs the desired task. The VV system has also been interfaced to the AL control system for the mechanical arms and has been tested on tasks that involve a combination of touch, force, and visual feedback.

+ AIM-296 CS-592
Robert Cartwright,
Practical Formal Semantic Definition and Verification Systems,
Thesis: Ph.D. in Computer Science,
158 pages, December 1976. Cost: $6.15

Despite the fact that computer scientists have developed a variety of formal methods for proving computer programs correct, the formal verification of a non-trivial program is still a formidable task. Moreover, the notion of proof is so imprecise in most existing verification systems, that the validity of the proofs generated is open to question. With an aim toward rectifying these problems, the research discussed in this dissertation attempts to accomplish the following objectives:

1. To develop a programming language which is sufficiently powerful to express many interesting algorithms clearly and succintly, yet simple enough to have a tractable formal semantic definition.

2. To completely specify both proof theoretic and model theoretic formal semantics for this language using the simplest possible abstractions.

3. To develop an interactive program verification system for the language which automatically performs as many of the straightforward steps in a verification as possible. The first part of the dissertation decribes the motivation for creating TYPED LISP, a variant of PURE LISP including a flexible data type definition facility allowing the programmer to create arbitrary recursive types. It is argued that a powerful data type definition facility not only simplifies the task of writing programs, but reduces the complexity of the complementary task of verifying those programs.

The second part of the thesis formally defines the semantics of TYPED LISP. Every function symbol defined in a program P is identified with a function symbol in a first order predicate calculus language Lp. Both a standard model Mp and a natural deduction system Np are defined for the language Lp. In the standard model, each function symbol is interpreted by the least call-by-value fixed-point of its defining equation. An informal meta-mathematical proof of the consistency of the model Mp and the deductive system Np is given.

The final part of the dissertation describes an interactive verification system implementing the natural deduction system Np.

The verification system includes:

1. A subgoaler which applies rules specified by the user to reduce the proof of the current goal (or theorem) to the proof of one or more subgoals.

2. A powerful simplifier which automatically proves many non-trivial goals by utilizing user-supplied lemmas as well as the rules of Np.

With a modest amount of user guidance, the verification system has proved a number of interesting, non-trivial theorems including the total correctness of an algorithm which sorts by successive merging, the total correctness of the McCarthy-Painter compiler for expressions, the termination of a unification algorithm and the equivalence of an iterative algorithm and a recursive algorithm for counting the leafs of a tree. Several of these proofs are included in an appendix.

- AIM-297 CS-610
Terry Winograd,
A Framework for Understanding Discourse,
24 pages, April 1977.

There is a great deal of excitement in linguistics, cognitive psychology, and artificial intelligence today about the potential of understanding discourse. Researchers are studying a group of problems in natural language which have been largely ignored or finessed in the mainstream of language research over the past fifteen years. They are looking into a wide variety of phenomena, and although results and observations are scattered, it is apparent that there are many interrelationships. While the field s not yet at a stage where it is possible to lay out a precise unifying theory, this paper attempts to provide a beginning framework for studying discourse. Its main goal is to establish a general context and give a feeling for the problems through examples and references. Its four sections attempt to:

Delimit the range of problems covered by the term "discourse."

Characterize the basic structure of natural language based on a notion of communication.

Propose a general approach to formalisms for describing the phenomena and building theories about them

Lay out an outline of the different schemas involved in generating and comprehending language

- AIM-298 CS-611 ADA046703
Zohar Manna, Richard Waldinger,
The Logic of Computer Programming,
90 pages, June 1977.

Techniques derived from mathematical logic promise to provide an alternative to the conventional methodology for constructing, debugging, and optimizing computer programs. Ultimately, these techniques are intended to lead to the automation of many of the facets of the programming process.

In this paper, we provide a unified tutorial exposition of the logical techniques, illustrating each with examples. We assess the strengths and limitations of each technique as a practical programming aid and report on attempts to implement these methods in experimental systems.

+ AIM-299 CS-614 ADA049760
Zohar Manna, Adi Shamir,
The Convergence of Functions to Fixedpoints of Recursive Definitions,
45 pages, May 1977. Cost: $2.95

The classical method for constructing the least fixedpoint of a recursive definition is to generate a sequence of functions whose initial element is the totally undefined function and which converges to the desired least fixedpoint.

This method, due to Kleene, cannot be generalized to allow the construction of other fixedpoints.

In this paper we present an alternate definition of convergence and a new [fixedpoint access] method of generating sequences of functions for a given recursive definition. The initial function of the sequence can be an arbitrary function, and the sequence will always converge to a fixedpoint that is "close" to the initial function. This defines a monotonic mapping from the set of partial functions onto the set of all fixedpoints of the given recursive definition.

- AIM-300 CS-617
Terry Winograd,
On some Contested Suppositions of Generative Linguistics about the Scientific Study of Language,
25 pages, May 1977.

This paper is a response to a recently published paper which asserts that current work in artificial intelligence is not relevant to the development of theories of language. The authors of that paper declare that workers in AI have misconstrued what the goals of an explanatory theory of language should be, and that there is no reason to believe that the development of programs which could understand language in some domain could contribute to the development of such theories. This paper concentrates on the assumptions underlying their view of science and language. It draws on the notion of "scientific paradigms" as elaborated by Thomas Kuhn, pointing out the ways in which views of what a science should be are shaped by unprovable assumptions. It contrasts the procedural paradigm (within which artificial intelligence research is based) to the currently dominant paradigm typified by the work of Chomsky. It describes the ways in which research in artificial intelligence will increase our understanding of human language, and through an analogy with biology, raises some questions about the plausibility of the Chomskian view of language and the science of linguistics.

+ AIM-301 CS-624 ADA044231
Lester Earnest, et. al.,
Recent Research in Computer Science,
118 pages, June 1977. Cost: $5.00

This report summarizes recent accomplishments in six related areas: (1) basic AI research and formal reasoning, (2) image understanding, (3) mathematical theory of computation, (4) program verification, (5) natural language understanding, and (6) knowledge based programming.

+ AIM-302 CS-630 ADA049761
Zohar Manna, Richard Waldinger
Synthesis: Dreams => Programs,
119 pages, October 1977. Cost: $5.05

Deductive techniques are presented for deriving programs systematically from given specifications. The specifications express the purpose of the desired program without giving any hint of the algorithm to be employed. The basic approach is to transform the specifications repeatedly according to certain rules, until a satisfactory program is produced. The rules are guided by a number of strategic controls. These techniques have been incorporated in a running program synthesis system, called DEDALUS.

Many of the transformation rules represent knowlede about the program's subject domain (e.g. numbers, lists, sets); some represent the meaning of the constructs of the specification language and the target programming language; and a few rules represent basic programming principles. Two of these principles, the conditional-formation rule and the recursion-formation rule, account for the introduction of conditional expressions and of recursive calls into the synthesized program. The termination of the program is ensured as new recursive calls are formed.

Two extensions of the recursion-formation rule are discussed: a procedure-formation rule, which admits the introduction of auxilliary subroutines in the course of the synthesis process, and a generalization rule, which causes

the specifications to be extended to represent a more general problem that is nevertheless easier to solve.

The techniques of this paper are illustrated with a sequence of examples of increasing complexity; programs are constructed for list processing, numerical computation, and sorting. These techniques are compared with the methods of "structured programming", and with recent work on "program transformation".

The DEDALUS system accepts specifications expressed in a high-level language, including set notation, logical quantification, and a rich vocabulary drawn from a variety of subject domains. The system attempts to transform the specifications into a recursive, LISP-like target program. Over one hundred rules have been implemented, each expressed as a small program in the QLISP language.

- AIM-303 CS-631 ADA050806
Nachum Dershowitz, Zohar Manna,
Inference Rules for Program Annotation,
46 pages, October 1977.

Methods are presented whereby an Algol-like program, given together with its specifications, may be documented automatically. This documentation expresses invariant relationships that hold between program variables at intermediate points in the program, and explains the actual workings of the program regardless of whether the program is correct. Thus this documentation can be used for proving the correctness of the program, or may serve as an aid in the debugging of an incorrect program.

The annotation techniques are formulated as Hoare-like inference rules which derive invariants from the assignment statements, from the control structure of the program, or, heuristically, from suggested invariants. The application of these rules is demonstrated by two examples which have run on our implemented system.

+ AIM-304 CS-632 ADA048684
Todd Wagner,
Hardware Verification,
Thesis: PhD in Computer Science,
102 pages, September 1977. Cost: $4.55

Methods for detecting logical errors in computer hardware designs using symbolic manipulation instead of digital simulation are discussed. A non-procedural register transfer language is proposed that is suitable for describing how a digital circuit should perform. This language can also be used to describe each of the components used in the design. Transformations are presented which should enable the designer to either prove or disprove that the set of interconnected components correctly satisfy the specifications for the overall system.

The problem of detecting timing anomalies such as races, hazards, and oscillations is addressed. Also explored are some interesting relationships between the problems of hardware verification and program verification. Finally, the results of using an existing proof checking program on some digital circuits are presented. Although the theorem proving approach is not very efficient for simple circuits, it becomes increasingly attractive as circuits become more complex. This is because the theorem proving approach can use complicated component specifications without reducing them to the gate level.

+ AIM-305 CS-633 ADA048660
William Faught,
Motivation and Intensionality in a Computer Simulation Model,
Thesis: Ph.D. in Computer Science,
104 pages, September 1977. Cost: $4.60

This dissertation describes a computer simulation model of paranoia. The model mimics the behavior of a patient participating in a psychiatric interview by answering questions, introducing its own topics, and responding to negatively-valued (e.g., threatening or shame-producing) situations.

The focus of this work is on the motivational mechanisms required to instigate and direct the modelled behavior.

The major components of the model are:

(1) A production system (PS) formalism accounting for the instigation and guidance of behavior as a function of internal (affective) and external (real-world) environmental factors. Each rule in the PS is either an action pattern (AP) or an interpretation pattern (IP). Both may have either affect (emotion) conditions, external variables, or outputs of other patterns as their initial conditions (left-hand sides). The PS activates all rules whose left-hand sides are true, selects the one with the highest affect, and performs the action specified by the right-hand side.

(2) A model of affects (emotions) as an anticipation mechanism based on a small number of basic pain-pleasure factors. Primary activation (raising an affect's strength) occurs when the particular condition for the affect is anticipated (e.g., anticipation of pain for the fear affect). Secondary activation occurs when an internal construct (AP, IP, belief) is used and its associated affect is processed.

(3) A formalism for intensional behavior (directed by internal models) requiring a dual representation of symbol and concept. An intensional object (belief) can be accessed either by sensing it in the environment (concept) or by its name (token). Similarly, an intensional action (intention) can be specified either by its conditions in the immediate environment (concept) or by its name (token).

Issues of intelligence, psychopathological modelling, and artificial intelligence programming are discussed. The paranoid phenomenon is found to be explainable as an extremely skewed use of normal processes. Applications of these constructs are found to be useful in AI programs dealing with error recovery, incompletely specified input data, and natural language specification of tasks to perform.

+ AIM-306 CS-639 ADA053175
Cordell Green, David Barstow,
On Program Synthesis Knowledge,
63 pages, November 1977. Cost: $3.45

This paper presents a body of program synthesis knowledge dealing with array operations, space reutilization, the divide and conquer paradigm, conversion from recursive paradigms to iterative paradigms, and ordered set enumerations. Such knowledge can be used for the synthesis of efficient and in-place sorts including quicksort, mergesort, sinking sort, and bubble sort, as well as other ordered set operations such as set union, element removal, and element addition. The knowledge is explicated to a level of detail such that it is possible to codify this knowledge as a set of program synthesis rules for use by a computer-based synthesis system. The use and content of this set of programming rules is illustrated herein by the methodical synthesis of bubble sort, sinking sort, quicksort, and mergesort.

+ AIM-307 CS-640 ADA053176
Zohar Manna and Richard Waldinger,
Structured Programming Without Recursion,
10 pages, December 1977. Cost: $2.00

There is a tendency in presentations of structured programming to avoid the use of recursion as a repetitive construct, and to favor instead the iterative loop constructs. For instance, in his recent book, "A Discipline of Programming," Dijkstra bars recursion from his exemplary programming language, declaring that "I don't like to crack an egg with a sledgehammer, no matter how effective the sledgehammer is for doing so."

In introducing an iterative loop, the advocates of structured programming advise that we first find an *invariant assertion* and a *termination function*, and then construct the body of the loop so as to reduce the value of the termination function while maintaining the

truth of the invariant assertion. The decision when to introduce a loop, and the choice of an appropriate invariant assertion and termination function, are not dictated by the method, but are left to the intuition of the programmer.

Unfortunately, at each stage in the derivation of a program, there are innumerable conditions and functions that could be adopted as the invariant assertion and termination function of a loop. With so many plausible candidates around, a correct selection requires an act of precognitive insight.

As an alternative, we advocate a method of loop formation in which the loop is represented as a recursive procedure rather than as an iterative construct. A recursive procedure is formed when a subgoal in the program's derivation is found to be an instance of a higher-level goal. The decision to introduce the new procedure, its purpose, and the choice of the termination function are all dictated by the structure of the derivation.

The directness of this *recursion-formation* approach stems from the use of recursion rather than iteration as a repetitive construct. Recursion is an ideal vehicle for systematic program construction; in avoiding its use, the advocates of structured programming have been driven to less natural means.

+ AIM-308 CS-641 ADA053184
David Barstow,
Automatic Construction of Algorithms,
Thesis: Ph.D. in Computer Science,
220 pages, December 1977. Cost: $7.85

Despite the wealth of programming knowledge available in the form of textbooks and articles, comparatively little effort has been applied to the codification of this knowledge into machine-usable form. The research reported here has involved the explication of certain kinds of programming knowledge to a sufficient level of detail that it can be used effectively by a machine in the task of constructing concrete implementations of abstract algorithms in the domain of symbolic programming.

Knowledge about several aspects of symbolic programming has been expressed as a collection of four hundred refinement rules. The rules deal primarily with collections and mappings and ways of manipulating such structures, including several enumeration, sorting and searching techniques. The principle representation techniques covered include the representation of sets as linked lists and arrays (both ordered or unordered), and the representation of mappings as tables, sets of pairs, property list markings, and inverted mappings (indexed by range element). In addition to these general constructs, many low-level programming details are covered (such as the use of variables to store values).

To test the correctness and utility of these rules, a computer system (called PECOS) has been designed and implemented. Algorithms are specified to PECOS in a high-level language for symbolic programming. By repeatedly applying rules from its knowledge base, PECOS gradually refines the abstract specification into a concrete implementation in the target language. When several rules are applicable in the same situation, a refinement sequence can be split. Thus, PECOS can actually construct a variety of different implementations for the same abstract algorithm.

PECOS has successfully implemented algorithms in several application domains, including sorting and concept formation, as well as algorithms for solving the reachability problem in graph theory and for generating prime numbers. PECOS's ability to construct programs from such varied domains suggests both the generality of the rules in its knowledge base and the viability of the knowledge-based approach to automatic programming.

+ AIM-309 CS-646
Nelson, C.G., Derek Oppen,
Efficient Decision Procedures Based on Congruence Closure,
15 pages, January 1978. Cost: $2.15

We define the notion of the *congruence closure*

of a relation on a graph and give a simple algorithm for computing it. We then give decision procedures for the quantifier-free theory of equality and the quantifier-free theory of LISP list structure, both based on this algorithm. The procedures are fast enough to be practical in mechanical theorem proving: each procedure determines the satisfiability of a conjunction of length n of literals in time O(n†2). We also show that if the axiomatization of the theory of list structure is changed slightly, the problem of determining the satisfiability of a conjunction of literals becomes NP-complete. We have implemented the decision procedures in our simplifier for the Stanford Pascal Verifier.

An earlier version of this paper appeared in the Proceedings of the 18th Annual Symposium on Foundations of Computer Science, Providence, October 1977.

+ AIM-310 CS-651 ADA058601
Nachum Dershowitz, Zohar Manna,
Proving Termination with Multiset Orderings,
33 pages, March 1978. Cost: $2.65

A common tool for proving the termination of programs is the well-founded set, a set ordered in such a way as to admit no infinite descending sequences. The basic approach is to find a termination function that maps the elements of the program into some well-founded set, such that the value of the termination function is continually reduced throughout the computation. All too often, the termination functions required are difficult to find and are of a complexity out of proportion to the program under consideration. However, by providing more sophisticated well-founded sets, the corresponding termination functions can be simplified.

Given a well-founded set S, we consider multisets over S, "sets" that admit multiple occurrences of elements taken from S. We define an ordering on all finite multisets over S that is induced by the given ordering on S. This multiset ordering is shown to be well-founded.

The value of the multiset ordering is that it permits the use of relatively simple and intuitive termination functions in otherwise difficult termination proofs. In particular, we apply the multiset ordering to provide simple proofs of the termination of production systems, programs defined in terms of sets of rewriting rules.

+ AIM-311 CS652
Greg Nelson, Derek C. Oppen,
Simplification by Cooperating Decision Prodcedures,
20 pages, April 1978. Cost: $2.25

We describe a simplifier for use in program manipulation and verification. The simplifier finds a normal form for any expression over the language consisting of individual variables, the usual boolean connectives, equality, the conditional function cond (denoting if-then-else), the numberals, the arithmetic functions and predicates +, - and ≤, the LISP constants, functions and predicates nil, car, cdr, cons and atom, the functions store and select for storing into and selecting from arrays, and uninterpreted function symbols. Individual variables range over the union of the reals, the set of arrays, LISP list structure and the booleans true and false.

The simplifier is complete; that is, it simplifies every valid formula to true. Thus it is also a decision procedure for the quantifier-free theory of reals, arrays and list structure under the above functions and predicates.

The organization of the simplifier is based on a method for combining decision procedures for several theories into a single decision procedure for a theory combining the original theories. More precisely, given a set S of functions and predicates over a fixed domain, a satisfiability program for S is a program which determines the satisfiability of conjunctions of literals (signed atomic formulas) whose predicate and function symbols are in S. We give a general procedure for combining satisfiability programs for sets S and T into a single satisfiability program for S ∪ T, given certain conditions on S and T.

The simplifier described in this paper is currently used in the Stanford Pascal Verifier.

+ AIM-312 CS-657 ADA065502
John McCarthy, Masahiko Sato, Takeshi Hayashi, Shigeru Igarashi,
On the Model Theory of Knowledge,
11 pages, April 1978. Cost: $2.00

Another language for expressing "knowing that" is given together with axioms and rules of inference and a Kripke type semantics. The formalism is extended to time-dependent knowlege. Completeness and decidability theorems are given. The problem of the wise men with spots on their foreheads and the problem of the unfaithful wives are expressed in the formalism and solved.

+ AIM-313 CS-660
Bruce E. Shimano,
The Kinematic Design and Force Control of Computer Controlled Manipulators,
Thesis: Ph.D. in Mechanical Engineering,
140 pages, March 1978. Cost: $5.65

The research presented in this dissertation concerns two problems involving mechanical manipulators which are operated under computer control: (1) the design and analysis of the structural parameters of manipulators with respect to their affect on work space, and (2) the real time control of manipulators using force feedback.

The positioning properties of mechanical manipulators are considered in terms of the motion of lines which are used to represent the manipulator's links and axes. A systematic method is developed for deriving equations which yield the normal distance, relative angle of twist, and moment between any two axes of rotation of a manipulator or between an axis of rotation and a link axis. This method is applied to the study of manipulators with at most three revolute axes, and conclusions are drawn concerning the dependence of the manipulator's range of motion on the dimensions of its links.

The possible number of extreme positions for a two link mechanism is studied, and it is shown how the selection of certain special link geometries affects the ability of the manipulator to maneuver around obstacles. In addition, a new geometric relationship between the extreme normal distance between the revolute axes os a manipulator and the positions of its intermediate axes is derived. This condition is applied to the development of an efficient means of determining the extreme distance positions given that the geometry of the manipulator is specified. It is also applied tothe reverse problem, i.e. of synthesizing the link dimensions of a manipulator given that one or more extreme distances is specified. Solutions to both problems are presented for manipulators containing an arbitrary number of links.

By interpreting the line analysis in terms of forces and torques, the previous results regarding displacements and changes in orientation are shown to be directly applicable to the analysis of joint and link force loads.

+ AIM-314 CS-678
Derek C. Oppen,
Reasoning About Recursively Defined Data Structures,
15 pages, July 1978. Cost: $2.15

A decision algorithm is given for the quantifier-free theory of recursively defined data structures which, for a conjunction of length n, decides its satisfiability in time linear in n. The first-order theory of recursively defined data structures, in particular the first-order theory of LISP list structure (the theory of CONS, CAR and CDR), is shown to be decidable but not elementary recursive.

+ AIM-315 CS-687 ADA065698
Richard W. Weyhrauch,
Prolegomena to a Theory of Formal Reasoning,
41 December 1978. This paper is an introduction to the mechanization of a theory of reasoning. Currently formal systems are out of favor with the AI community. The aim of this paper is to explain how formal systems can

be used in AI by explaining how traditional ideas of logic can be mechanized in a practical way. The paper presents several new ideas. Each of these is illustrated by giving simple examples of how this idea is mechanized in the reasoning system FOL. That is pages, this is not just theory but there is an existing running implementation of these ideas. Cost: $2.85

In this paper: 1) we show how to mechanize the notion of model using the idea of a simulation structure and explain why this is particularly important to AI, 2) we show how to mechanize the notion of satisfaction, 3) we present a very general evaluator for first order expressions, which subsumes PROLOG and we propose as a natural way of thinking about logic programming, 4) we show how to formalize metatheory, 5) we describe reflection principles, which connect theories to their metatheories in a way new to AI, 6) we show how these ideas can be used to dynamically extend the strength of FOL by "implementing" subsidiary deduction rules, and how this in turn can be extended to provide a method of describing and proving theorems about heuristics for using these rules, 7) we discuss one notion of what it could mean for a computer to learn and give an example, 8) we describe a new kind of formal system that has the property that it can reason about its own properties, 9) we give examples of all of the above.

- AIM-316 CS-671
Jerrold M. Ginsparg,
Natural Language Processing in an Automatic Programming Domain,
Thesis: Ph.D. in Computer Science,
172 pages, June 1978.

This paper is about communicating with computers in English. In particular, it describes an interface system which allows a human user to communicate with an automatic programming system in an English dialogue.

The interface consists of two parts. The first is a parser called Reader. Reader was designed to facilitate writing English grammars which are nearly deterministic in that they consider a very small number of parse paths during the processing of a sentence. This efficiency is primarily derived from using a single parse structure to represent more than one syntactic interpretation of the input sentence.

The second part of the interface is an an interpreter which represents Reader's output in a form that can be used by a computer program without linguistic knowledge. The Interpreter is repsonsible for asking questions of the user, processing the user's replies, building a representation of the program the user's replies describe, and supplying the parser with any of the contextual information or general knowledge it needs while parsing.

+ AIM-317 CS-675
Donald E. Knuth,
Tau Epsilon Chi, A System for Technical Text,
200 pages, September 1978. Cost: $7.30

This is the user manual for TEX, a new document compiler at SU-AI, intended to be an advance in computer typesetting. It is primarily of interest to the Stanford user community and to people contemplating the installation of TEX at their computer center.

+ AIM-318 CS-688
Zohar Manna,
Six Lectures on the Logic of Computer Programming,
54 pages, November 1978. Cost: $3.20

These are notes from six tutorial lectures given at the NSF Regional Conference, Renssellaer Polytechnic Institute, May 29-June 2, 1978. The lectures deal with various aspects of the logic of computer programming: partial correctness of programs, termination of programs, total correctness of programs, systematic program annotation, synthesis of programs and termination of production systems.

+ AIM-319 CS-689
Charles G. Nelson,
An $n^{\log n}$ algorithm for the two-variable-per-constraint linear programming satisfiability problem,
20 pages, December 1978. Cost: $2.25

A simple algorithm is described which determines the satisfiability over the reals of a conjunction of linear inequalities, none of which contains more than two variables. In the worst case the algorithm requires time $O(mn^{\lfloor \log 2n \rfloor + 3} \log n)$, where n is the number of variables and m the number of inequalities. Several considerations suggest that the algorithm may be useful in practice: it s simple to implement, it is fast for some importat special cases, and if the inequalities are satisfiable it provides valuable information about their solution set. The algorithm is particularly suited to applications in mechanical program verification. This is also known as Stanford Verification Group Report 10.

+ AIM-320 CS-690 ADA065558
Zohar Manna,
A Deductive Approach to Program Synthesis,
30 pages, December 1978. Cost: $2.55

Program synthesis is the systematic derivation of a program from given specifications. A deductive approach to program synthesis is presented for the construction of recursive programs. This approach regards program synthesis as a theorem-proving task and relies on a theorem-proving method that combines the features of transformation rules, unification, and mathematical induction within a single framework.

+ AIM-321 CS-695 ADA06562
McCarthy, et.al,
Recent Research in Artificial Intelligence and Programming Methodology,
94 pages, November 1978. Cost: $4.35

Summarizes recent research in the following areas:

artificial intelligence and formal reasoning,
mathematical theory of computation and program synthesis,
program verification,
image understanding,
knowledge based programming.

+ AIM-322 CS-716
Georgeff, Michael,
A Framework for Control in Production Systems,
35 pages, January 1979. Cost: $2.70

A formal model for representing control in production systems is defined. The formalism allows control to be directly specified independently of the conflict resolution scheme, and thus allows the issues of control and nondeterminism to be treated separately. Unlike previous approaches, it allows control to be examined within a uniform and consistent framework.

It is shown that the formalism provides a basis for implementing control constructs which, unlike existing schemes, retain all the properties desired of a knowledge based system -- modularity, flexibility, extensibility and explanatory capacity. Most importantly, it is shown that these properties are not a function of the lack of control constrains, but of the type of information allowed to establish these constraints.

Within the formalism it is also possible to provide a meaningful notion of the power of control constructs. This enables the types of control required in production systems to be examined and the capacity of various schemes to meet these requirements to be determined.

Schemes for improving system efficiency and resolving nondeterminism are examined, and devices for representing such meta-level knowledge are described. In particular, the objectification of control information is shown to provide a better paradigm for problem solving and for talking about problem solving. It is also shown that the notion of control provides a basis for a theory of transformation of production systems, and that this provides a

uniform and consistent approach to problems involving subgoal protection.

+ AIM-323 CS-718 Mujtaba Shahid Ron Goldman,
AL Users' Manual,
136 pages, January 1979. Cost: $5.50

This document describes the current state of the AL system now in operation at the Stanford Artificial Intelligence Laboratory, and teaches the reader how to use it. The system consists of AL, a high-level programming language for manipulator control useful in industrial assembly research; POINTY, an interactive system for specifying representation of parts; and ALAID, an interactive debugger for AL.

+ AIM-324 CS-717
Cartwright, Robert, John McCarthy,
Recursive Programs as Functions in a First Order Theory,
32 pages, March 1979. Cost: $2.60

Pure Lisp style recursive function programs are represented in a new way by sentences and schemata of first order logic. *This permits easy* and natural proofs of extensional properties of such programs by methods that generalize structural induction. It also systematizes known methods such as *recursion induction, subgoal induction, inductive assertions* by interpreting them as first order axiom schemata. We discuss the metatheorems justifying the representation and techniques for proving facts about specific programs. We also give a simpler version of the Goedel-Kleene way of representing computable functions by first order sentences.

+ AIM-325 CS-724
McCarthy, John,
First Order Theories of Individual Concepts and Propositions,
19 pages, March 1979. Cost: $2.25

We discuss first order theories in which individual concepts are admitted as mathematical objects along with the things that reify them. This allows very straightforward formalizations of knowledge, belief, wanting, and necessity in ordinary first order logic without modal operators. Applications are given in philosophy and in artificial intelligence.

+ AIM-326 CS-725
McCarthy, John,
Ascribing Mental Qualities to Machines,
25 pages, March 1979. Cost: $2.40

Ascribing mental qualities like beliefs, intentions and wants to a machine is sometimes correct if done conservatively and is sometimes necessary to express what is known about its state. We propose some new definitional tools for this: definitions relative to an approximate theory and second order structural definitions.

+ AIM-327 CS-727
Filman, Robert Elliot,
The Interaction of Observation and Inference,
Thesis: PhD in Computer Science,
232 pages, April 1979. Cost: $8.20

An intelligent computer program must have *both a representation* of its knowledge, and a mechanism for manipulating that knowledge in a reasoning process. This thesis is an examination of the problem of formalizing the expression and solution of reasoning problems in a machine manipulable form. It is particularly concerned with analyzing the interaction of the standard form of deductive steps with an observational analogy obtained by performing computation in a semantic model.

Consideration in this dissertation is centered on the world of retrograde analysis chess, a particularly rich domain for both observational tasks and long deductive sequences.

A formalization is embodied in its axioms, and a major portion of this dissertation is devoted to both axiomatizing the rules of chess, and discussing and comparing the representational decisions involved in that axiomatization. Consideration was given to not only the necessity for these particular choices (and

possible alternatives) but also the implications of these results for designers of representational systems for other domains.

Using a reasoning system for first order logic, "FOL", a detailed proof of the solution of a difficult retrograde chess puzzle was constructed. The close correspondence between this "formal" solution to the problem, and an "informal, descriptive" analysis a human might present was shown.

The proof and axioms were then examined for their relevance to general epistemological formalisms. The importance of several different mechanisms were considered. These included: 1) retaining both the notion of "current status" (typically embodied as the current chessboard) and that of "historical state" (a hypothetical game played to reach desired place), 2) evaluating functional and predicate objects in the semantic model (the chess eye), 3) the value of "induction schemas" as partial solutions to frame problems, 4) the retention of explicit undefined elements within the representation, 5) the importance of manipulating multiple representations of objects, and 6) a comparison of state vector and modal representations.

+ AIM-328 CS-743
Bulnes-Rozas, Juan Bautista,
GOAL: A Goal Oriented Command Language for Interactive Proof Constructions,
Thesis: PhD in Computer Science,
178 June 1979. This thesis represents a contribution to the development of practical computer systems for interactive construction of formal proofs. Beginning with a summary of current research in automatic theorem proving pages, goal oriented systems, proof checking, and program verification, this work aims at bridging the gap between proof checking and theorem proving. Cost: $6.70

Specifically, it describes a system GOAL for the First Order Logic proof checker FOL. GOAL helps the user of FOL in the creation of long proofs in three ways: 1) as a facility for structured, top down proof construction; 2) as a semi-automatic theorem prover; and 3) as an extensible environment for the programming of theorem proving heuristics.

In GOAL, the user defines top level goals. These are then recursively decomposed into subgoals. The main part of a goal is a well formed formula that one desires to prove, but they include assertions, simplification sets, and other information. Goals can be tried by three different types of elements: *matchers*, *tactics*, and *strategies*.

The *matchers* attempt to prove a goal directly -that is without reducing it into subgoals- by calling decision procedures of FOL. Successful application of a matcher causes the proved goal to be added to the FOL proof.

A *tactic* reduces a goal into one or more subgoals. Each tactic is the inverse of some inference rule of FOL; the goal structure records all the necessary information so that the appropriate inference rule is called when all the subgoals of a goal are proved. In this way the goal tree unwinds automatically, producing a FOL proof of the top level goal from the proofs of its leaves.

The *strategies* are programmed sequences of applications of tactics and matchers. They do not interface with FOL directly. Instead, they simulate a virtual user of FOL. They can call the tactics, matchers, other strategies, or themselves recursively. The success of this approach to theorem proving success is documented by one heuristic strategy that has proved a number of theorems in Zermelo-Fraenkel Axiomatic Set Theory. Analysis of this strategy leads to a discussion of some trade offs related to the use of *assertions* and *simplification sets* in goal oriented theorem proving.

The user can add new tactics, matchers, and strategies to GOAL. These additions cause the language to be extended in a uniform way. The description of new strategies is done easily, at a fairly high level, and no faulty deduction is possible. Perhaps the main contribution of

GOAL is a high level environment for easy programming of new theorem proving applications in the First Order Predicate Calculus.

The thesis ends with two appendixes presenting complete proofs of Ramsey's theorem in axiomatic Set Theory and of the correctness of the Takeuchi function.

(It is planned that both FOL and GOAL will be made available over the ARPANET this year. Inquiries regarding their use should be addressed to Dr. R. Weyhrauch at the Stanford Artificial Intelligence Laboratory, SU-AI).

+ AIM-329 CS-747 AD-A076-872
David Edward Wilkins,
Using Patterns and Plans to Solve Problems and Control Search,
Thesis: PhD in Computer Science,
264 pages, June 1979. Cost: $9.10

The type of reasoning done by human chess masters has not been done by computer programs. The purpose of this research is to investigate the extent to which knowledge can replace and support search in selecting a chess move and to delineate the issues involved. This has been carried out by constructing a program, PARADISE (PAtternRecognition Applied to DIrecting SEarch), which finds the best move in tatically sharp middle game positions from the games of chess masters.

PARADISE plays chess by doing a large amount of static, knowledge-based reasoning and constructing a small search tree (tens of hundreds of nodes) to confirm that a particular move is best, both characteristics of human chess masters. A "Production-Language" has been developed for expressing chess knowledge in the form of productions, and the knowledge base contains about 200 productions written in this language. The actions of the rules post concepts in the data base while the conditions match patterns in the chess position and data base. The patterns are complex to match (search may be involved). The knowledge base was built incrementally, relying on the system's ability to explain its reasoning and the ease of writing and modifying productions. The productions are structured to provide "concepts" to reason with, methods of controlling pattern instantiation, and means of focusing the system's attention on relevant parts of the knowledge base. PARADISE knows why it believes its concepts and may reject them after more analysis.

PARADISE uses its knowledge base to control the search, and to do a static analysis of a new position which produces concepts and plans. Once a plan is formulated, it guides the tree search for several ply by focussing the analysis at each node on a small group of productions. Expensive static analyses are rarely done at new positions created during the search. Plans may be elaborated and expanded as the search proceeds. These plans (and the use made of them) are more sophisticated than those in previous AI systems. Through plans, PARADISE uses the knowledge applied during a previous analysis to control the search for many nodes.

By using the knowledge base to help control the search, PARADISE has developed an efficient best-first search which uses different startegies at the top level. The value of each move is a range which is gradually narrowed by doing best-first searches until it can be shown that one move is best. By using a global view of the search tree, information gathered during the search, and information produced by static analyses, the program produces enough terminations to force convergence of the search. PARADISE does not place a depth limit on the search (or any other artificial effort limit). The program incorporates many cutoffs that are not useful in less knowledge-oriented programs (e.g., restrictions are placed on concatenation of plans, accurate forward prunes can be made on the basis of static analysis results, a causality facility determines how a move can affect a line already searched using patterns generated during the previous search).

PARADISE has found combinations as deep as

19 ply and performs well when tested on 100 standard problems. The modifiability of the knowledge base is excellent. Developing a search strategy which converges and using a large knowledge base composed of patterns of this complexity to achieve expert performance on such a complex problem is an advance on the state of the art.

+ AIM-330 CS-751
Zohar Manna, Amir Pnueli,
The Modal Logic of Programs,
36 pages, September 1979. Cost: $2.70

We explore the general framework of Modal Logic and its applicability to program reasoning. We relate the basic concepts of Modal Logic to the programming environment: the concept of "world" corresponds to a program state, and the concept of "accessibility relation" corresponds to the relation of derivability between states during execution. Thus we adopt the Temporal interpretation of Modal Logic. The variety of program properties expressible within the modal formalism is demonstrated.

The first axiomatic system studied, the *sometime system*, is adequate for proving total correctness and 'eventuality' properties. However, it is inadequate for proving invariance properties. The stronger *nexttime system* obtained by adding the *next* operator is shown to be adequate for invariances as well.

+ AIM-331 CS-755
Elaine Kant,
Efficiency Consideration in Program Synthesis: A Knowedge-Based Approach,
Thesis: PhD in Computer Science,
160 pages, July 1979. Cost: $6.20

This thesis presents a framework for using efficiency knowledge to guide program synthesis when stepwise refinement is the primary synthesis technique. The practicality of the framework is demonstrated via the implementation of a particular search-control system called LIBRA. The framework is also one model for how people write and analyze programs.

LIBRA complements an interactive program-synthesis project by adding efficiency knowledge to the basic coding knowledge and by automating the implementation-selection process. The program specifications include size and frequency notes, the performance measure to be minimized, and some limits on the time and space of the synthesis task itself. LIBRA selects algorithms and data representations and decides whether to use "optimizing" transformations by applying knowledge about time and storage costs of program components. Cost estimates are obtained by incremental, algebraic program analysis.

Some of the most interesting aspects of the system are explicit rules about plausible implementations, constraint setting, multiple-exit-loop analysis, the comparison of optimistic and achievable cost estimates to identify important decisions, and resource allocation and the basis of importance. LIBRA has automatically guided the construction of programs that classify, retrieve information, sort, and computer prime numbers.

+ AIM-332 CS-762 AD-A083-229
Donald Knuth,
METAFONT: A System for Alphabet Design,
110 pages, September 1979. Cost: $4.80

This is the user's manual for METAFONT, a companion to the TEX typesetting system. The system makes it fairly easy to define high quality fonts of type in a machine-independent manner; a user writes "programs" in a new language developed for this purpose. By varying parameters of a design, an unlimited number of typefaces can be obtained from a single set of programs. The manual also sketches the algorithms used by the system to draw the character shapes.

+ AIM-333 CS-772
Brian P. McCune,
Building Program Models Incrementally from Informal Descriptions,
Thesis: PhD in Computer Science,
146 pages, October 1979. Cost: $5.80

Program acquisition is the transformation of a program specification into an executable, but not necessarily efficient, program that meets the given specification. This thesis presents a solution to one aspect of the program acquisition problem: the incremental construction of program models from informal descriptions. The key to the solution is a framework for incremental program acquisition that includes (1) a formal language for expressing program fragments that contain informalities, (2) a control structure for the incremental recognition and assimilation of such fragments, and (3) a knowledge base of rules for acquiring programs specified with informalities.

The thesis describes a LISP based computer system called the *Program Model Builder* (abbreviated "PMB"), which receives informal program fragments incrementally and assembles them into a very high level program model that is complete, semantically consistent, unambiguous, and executable. The program specification comes in the form of partial *program fragments* that arrive in any order and may exhibit such informalities as inconsistencies and ambiguous references. Possible sources of fragments are a natural language parser or a parser for a surface form of the fragments. PMB produces a program model that is a complete and executable computer program. The *program fragment language* used for specifications is a superset of the language in which program models are built. This *program modelling language* is a very high level programming language for symbolic processing that deals with such information structures as sets and mappings.

PMB has expertise in the general area of simple symbolic computations, but PMB is designed to be independent of more specific programming domains and particular program specification techniques at the user level. However, the specifications given to PMB must still be algorithmic in nature. Because of the very high level nature of the program model produced, PMB also operates independently from implementation details such as the target computer and low level language. PMB has been tested both as a module of the PSI program synthesis system and independently. Models built as part of PSI have been acquired via natural language dialogs and execution traces and have been automatically coded into LISP by other PSI modules. PMB has successfully built a number of moderately complex programs for symbolic computation.

By the design the user is allowed to have control of the specification process. Therefore PMB must handle program fragments interactively and incrementally. Interesting problems arise because these informal fragments may arrive in an arbitrary order, may convey an arbitrarily small amount of new information, and may be incomplete, semantically inconsistent, and ambiguous. To allow the current point of focus to change, a *program reference language* has been designed for expressing patterns that specify what part of the model a fragment refers to. Various combinations of syntactic and semantic reference patterns in the model may be specified.

The recognition paradign used by PMB is a form of subgoaling that allows the parts of the program to be specified in an order chosen by the user, rather than dictated by the system. Knowledge is represented as a set of data driven antecedent rules of two types, *response rules* and *demons*, which are triggered respectively by either the input of new fragments or changes in the partial program model. In processing a fragment, a response rule may update the partial program model and create new subgoals with associated response rules. To process subgoals that are completely internal to PMB, demon rules are created that delay execution until their prerequisite information in the program model has been filled in by response rules or perhaps other demons.

PMB has a knowledge base of rules for handling modelling language constructs, processing informalities in fragments, monitoring the consistency of the model, and

transforming the program to canonical form. Response rules and simple demons are procedural. *Compond demons* have more complex antecedents that test more than one object in the program model. Compond demons are declarative antecedent patterns that are expanded automatically into procedural form.

+ AIM-334 CS-788
John McCarthy,
CIRCUMSCRIPTION – A Form of Non-Monotonic Reasoning,
15 pages, February 1980. Cost: $2.15

Humans and intelligent computer programs must often junp to the conclusion that the objects they can determine to have certain properties or relations are the only objects that do. *Circumscription* formalizes such conjectural reasoning.

+ AIM-335 CS-796
Arthur Samuel,
Essential E,
33 pages, March 1980. Cost: $2.65

This is an introductory manual describing the display-oriented text editor E that is available on the Stanford A.I. Laboratory PDP-10 computer. The present manual is intended to be used as an aid for the beginner as well as for experienced computer users who either are unfamiliar with the E editor or use it infrequently. Reference is made to the two on-line manuals that help the beginner to get started and that provide a complete description of the editor fr the experienced user.

E is commonly used for writing computer programs and for preparing reports and memoranda. It is not a document editor, although it does provide some facilities for getting a document into a pleasing format. The primary emphasis is that of speed, both in terms of the number of key strokes required of the user and in terms of the demands made on the computer system. At the same time, E is easy to learn and it offers a large range of facilities that are not available on many editors.